Design and Engineering, in Spite of Open-Source

Aurélien Pierre

MANUSCRIPT, VERSION 2

Free online edition — first version: August 2, 2026

Last updated: August 28, 2026

editions.aurelienpierre.com/en/concevoir/

A tool serves its user, or it betrays them. There is no in-between.

This book does two things. It documents — records open, everything counted, dated, verifiable — how one of the best-known free-software projects in photography treats its users, its contributors, and its critics. And it lays out the method this project wanted nothing to do with: state the problem before the solution, listen without obeying, measure without lying to yourself, refuse what you will not be able to maintain.

I was there. I wrote some of its modules, I left it, I forked it. Everything that could be measured has been; the rest is given for what it is: my judgment, signed.

The conclusion uses a word engineering never utters: to design is to care. And free software, which entrusted everyone's tools to those who play, cannot remain the plaything of the few who can, at the expense of the majority who need.

Prologue — Three paths

The writing of this book took 3 different paths to reach the result you now have before you. The first path was to compile, synthesize and deepen some fifteen of my articles on design, plus 50 minutes of voice notes, accumulated over the past decade and rooted in my research and development experience in free and open-source software, and to use the opportunity to enrich or qualify my empirical observations with what the scientific literature had to say on the same subjects. Designers, even non-engineers in general and non-programmers in particular, should therefore find in the first parts methods for analyzing problems and identifying solutions that apply to many fields.

The next path was training my successor to maintain Ansel, a photo-processing software project I forked from Darktable in 2022, after the umpteenth massive overhaul of the user interface, made at the expense of usability and without any defined need or problem, and which, for me, marked the final fracture between the development team and my work ethic. I had long said that if I ever had to teach a design course, the Darktable project would be my textbook case of every example not to reproduce, so thoroughly were bad decisions taken everywhere it was possible to take them. That judgment can be measured, and this book measures it on four records: user support and user engagement, which are collapsing; contributions, which are declining; bug fixing, which no longer keeps up; code complexity, quantified

with standard metrics, which is soaring. And these decisions were not unlucky bets: releasing on a fixed schedule with no paid staff to absorb the pace, merging faster than a human can review, accepting without criteria, piling up options rather than deciding, letting power concentrate without mandate or appeal — all choices that the literature, in software engineering as in organization science, had already filed among the bad ideas, decades before they were made. It is therefore important to document these errors, by their effects as much as by this contempt for established knowledge, so as not to condemn ourselves to repeating them. They are at once the fruit of an ignorance of what I will present here (industrial design methods and the psycho-social structures of the productive apparatus), and of an absence of interest in the Other, that is, whoever is not "Us" (the social typology of this "Us" and its codes will be detailed). It is important to understand precisely why these errors were errors, and not mere accidents, and in which strata of the production apparatus they occurred, because blindly applying the diametrically opposite strategy would only generate a different kind of error: the space of bad solutions is infinite. I do not want Ansel to become another Darktable.

The last path is the story of a solitude that turns into a plea: that of a developer who programmed to solve the problems of humans facing the machine, in the middle of a community that programs because it is fun, and that gets equally enthusiastic about any technological innovation, without ever asking whether the novelty will make its work better and easier. Worse, it never asks how what amuses Us can become suffering for the Other. For this

community defines itself first by its education and its privileges, and therefore by the needs it can afford to ignore, all while maintaining a double discourse of emancipation and digital sovereignty that serves only to silence the user's needs (and complaints).

The conclusion of this book wrote itself, chapter after chapter, without my noticing: design done well — engineering in the service of the human — must take away from medicine its exclusive claim on care. Confining engineering to technical problems is myopia, for there is no technical problem without a *technicized* human left helpless before a machine, or before a problem of such scale that the machine's productivity becomes the only realistic path. The technicized human is a human to be cared for, because technology can relieve suffering just as well as it can create new suffering, and worse. We no longer hesitate to frame the engineer's responsibility in terms of social and environmental impact, but these are broad concepts, shapeless objects that commit you only to producing vague metrics proving the question was studied, and then to carrying on exactly as before while telling yourself that progress is a train moving by its own inevitable nature, whether you choose to board it or not. No: care is the relation of the one who can over the one who needs, and that asymmetry creates the power we try to hide beneath the technical appearance of the problem, because it is at bottom a relation of domination of one human over another. Nothing less, nothing more.

And that is precisely what is wrong with free software: a circle of privileged people building tools for that same circle, while

claiming to produce universal commons, through vaguely libertarian and falsely Marxist concepts that serve only to convince those excluded from the circle that they owe their exclusion to their personal characteristics, to their internal flaws. When it is the structure of the system that produces its own excluded.

But this care, to be just, cannot ignore the one who commits his labor, his time and his energy to serving the needs of others; nor is it a matter of turning it into the oppression and exploitation of the one who must produce by those who want to use, all the more so since producers are always scarcer than users, and the crowd is a mass one must also protect oneself from. One might be tempted to see this as a rebalancing of power, but it is only a vicious circle that merely legitimizes a two-way violence. And this is another failing of free software, which in truth springs from the same problem: thinking about the social structure in which free software emerges is the blind spot of the free-software movement, which has focused all its attention on licensing problems — that is, on intellectual property — and nothing else.

The thesis of this book holds in 4 points:

1. Design is a discipline for solving a human problem that we have chosen to read as technical — and the object of design is never just the machine, but the whole system: the engine, the interface, the documentation, the community, the economy that carries it, and the worker who maintains it all.
2. Free software must stop being the playground of a socially homogeneous club and become a public service that meets

its users' needs with benevolence, all the more so since its rhetoric already plays both sides.
3. Design in a public service then becomes an act of care, and care induces an asymmetric relation that requires a strict code of ethics if it is not to become an act of oppression within a structure of domination.
4. Guaranteeing that this code of ethics is respected, arbitrating between users' needs, between workers' needs, and guaranteeing safe working conditions cannot be contemplated outside an organized social structure, one that correctly identifies positions of power, holds them accountable, chooses the individuals it places there and for how long.

These three paths have left a book with two sources. The first is the literature: this book borrows heavily. It invents no law: it applies to a field that had escaped them — the design of free software — results established elsewhere, often long ago.

Running beneath everything, **epistemology**, because it asks the crucial questions and already offers answers to them: what is the difference between knowledge and belief; how to reconcile subjectivities with the expected objectivity; how do we know what we know, and how much credit to give it. The ready-made answers we will reuse are the reproducibility of results — the very norm of the experimental method —, Popper's falsifiability criteria, the integrity Feynman opposed to "cargo cult science", and the *tacit* part of knowledge that Polanyi named, the part no rule can formalize.

For the design approach, we rely on the **design sciences** — Simon, for whom to design is to turn an existing situation into a preferred one; Rittel and Cross, for what separates a design problem from an ordinary problem. **Engineering** adds its methods, proven on physical objects: Pahl and Beitz's systematic design, Pugh's concept selection, Ulrich and Eppinger's product development; the **industrial culture of quality and continuous improvement** — root-cause analysis, the Ishikawa diagram, Toyota's "five whys", *lean* waste elimination; and **reliability and safety engineering**, which teaches you to design so that things hold and fail cleanly (Reason, Leveson, Dekker, Hollnagel).

On the user's side, **human factors, ergonomics and human-computer interaction** provide the laws: the average user who does not exist (Daniels's anthropometry), the cost of choice (Hick), the error that is almost always a design error (Norman), the contextual inquiry that goes to see the real work instead of assuming it. And to understand how this user perceives, chooses and errs — and what competence actually is —, **cognitive and behavioral psychology** (from White and McClelland to decision biases) and **metacognition**, that discipline of thinking against oneself.

For the collective — the power that hides where we believe it abolished, the silence that organizes itself, the mediocrity that settles in —, we borrow from the **sociology of work and organizations**: Michels's iron law, Freeman's tyranny of structurelessness, organizational silence, Merton's obliteration by incorporation, Becker's labeling. For what the milieu produces,

and for whom, from the **sociology and philosophy of technology**: Akrich's *I-methodology*, Bourdieu's habitus, Mumford's distinction between the tool and the machine, Ellul's critique of technique.

For the economics of this world without prices, from the **economics of open-source and the commons** (Lerner and Tirole, Eghbal, Hirschman, Ostrom) and from **behavioral economics and decision theory**: the sunk-cost fallacy, Kahneman and Tversky's loss aversion, Goodhart's and Campbell's laws on measures that become targets, the protected values people refuse to trade off.

For including the user in the design chain, from the **ethics of care** — Tronto, Fisher, Gilligan — and from Illich's philosophy of conviviality. And for the analysis of the mechanisms of closure and censorship — how a substantive disagreement gets requalified as a fault of tone, how anger gets discredited as hysteria —, from **feminist philosophy and critical theory**: Fricker's epistemic injustice, Audre Lorde's words on the master's tools.

That is the first source of this book: the literature, what others have established elsewhere, on other terrains. It has a second, of an entirely different nature — the reader knows it already: the Darktable case, the one from the second path. Theory states laws; the Darktable case shows what it costs to violate them repeatedly. This source has nothing of the first, for it is neither neutral nor objective, but everything I assert is verifiable, having played out in public, in places that kept archives.

A case like this is worth two things, and the second weighs more than the first. The first: no better-documented case exists. Ten years of code-repository history, two forums queryable through their APIs, the log of a chat room, a bug tracker, my own user surveys published with their data and their code — all of it is public, and every claim, from either camp, can be checked against the record. The second is what a stranger, with the same archives before his eyes, would not have: I know where to look, and how to read. I know which thread matters, which commit tipped everything over, what a silence on a channel means, which innocuous decision was masking another. The archives are open to everyone; their meaning is open only to those who lived inside them. That is what I bring: situated knowledge, which its very situation makes at once sharper and less neutral.

A source like this — empirical, partial, interested — is usable on one condition only: that the method that processes it be stated. It holds in 4 rules. *The literature first*: every mechanism invoked was described decades before this case, on other terrains; the case illustrates known laws, it founds none. *Falsification*: what the community claims, I test against the records, and I publish the result both ways: the measures that go up appear in the same charts as those that go down, and every caption carries its own caveats. *Symmetry*: my messages, my reports and my commits are excluded from the counts whenever the question concerns the state of the house and not my own; and my side of the conflict is entered into the record, dated, with the same instruments as everyone else's. *The roles*: apart from public authors cited for their public writings, everyone appears here by their function —

the maintainer, the moderator, the author of an overhaul —, never to be singled out.

Everything I put forward from this case, I call *the record*. The word does not mean a private archive: the record is the set of exhibits cited in the notes throughout the chapters — forum threads, bug reports, code history, chat logs —, each dated and sourced, and, for everything the web allows to be archived, saved to a public archive at the date of consultation. The case is bounded by design — it occupies the parts "The collective" and "Freedom and its conditions", and the method chapters rest on the literature and on examples anyone can redo. Its function, though, runs through the whole book. What happened there is not an accident of persons: it is the natural trajectory of a project carried by the ordinary habitus, beliefs and ethos of this ecosystem — and most free-software projects look more like this one than like the salaried giants the literature studies. The book shows what these beliefs cost, counts the damage on public records, and defends, against them, values that put the user at the center. I document this trajectory exhibit by exhibit so that Ansel does not reproduce it, and so that any project that wants to avoid it can recognize it in time.

Free software is a workplace where no one hires the team, where no one answers for what they ship. When a decision does damage, there is no hierarchy to appeal to, no performance review, no dismissal — all that remains is public discussion, which turns poisonous precisely because it is the only lever. I used that lever: what had to be said was said, in the most exact terms, while I endured the decisions and cleaned up their damage; my

anger was legitimate and well-founded, and reducing a speech to its form in order to invalidate its substance is precisely one of the mechanisms we will study. This book draws the useful conclusion: when a small number keeps its hands on the same decisions for years on end, with no one having mandated them and no one able to revoke them, the problem goes beyond individuals — it is the system that makes such locking possible.

The reading contract follows from all of the above. This book is opinionated: I believe a design can be called good or bad, and I will say so. Whatever is measured or established carries its source in a note; everything else is my judgment, written in the first person. The book is no less falsifiable for it: when the literature contradicts my position on the same object, it is cited before it is contradicted.

This book, finally, is not only about software: it is about tools for work and tools for artistic creation — those objects designed to serve someone else's intent and purpose. Software is its main case study, because it is my terrain and because its fluid interfaces give the designer more degrees of freedom than any physical object — enough freedom to do very well, and very badly. But a paintbrush, a bridge or an administrative form obey the same laws: they serve their user's intent, or they betray it.

Part I — What designing means

1. The word and the thing

French has dealt the word *design* a curious fate: it borrowed it from English and kept only its third meaning, the least interesting one. In French, design is a matter of shapes — a couch is *design*, a kettle has *design* — and people happily recite that design is "when form meets function." English says something else. There, *design* descends from the *disegno* of the Italian workshops, which named at once the drawing and the intent: the line and the purpose. *To design* is to project — to fix a goal, then organize the means of reaching it. What English calls *design*, the engineer's French calls **conception**[1], and that is what this book is about. The drawing of shapes is only a distant consequence of it, and not the most important one.

This confusion of vocabulary is not an etymological footnote: it decides who feels entitled to speak. As long as design is a matter of shapes, everyone has eyes, so everyone has an opinion, and all opinions are equal. The moment design means designing — a stated problem, constraints, criteria, a solution that can defend itself — opinion stops being enough: you need reasons. Most of the sterile discussions about a piece of software's design come down to this slippage of meaning, and the first task is therefore to pin the word down.

Herbert Simon gave the thing its broadest and most useful definition: "Everyone designs who devises courses of action aimed at changing existing situations into preferred ones."[2] Each of his terms matters. *Existing situations*: design starts from a state of the world, not from a blank page or a whim. *Preferred*: there is a goal, hence someone who prefers — a human whose intention is the measure of success. *Courses of action*: design produces an organized path, not an inspiration. Under this definition, the engineer designs, but so does the physician building a protocol, the legislator drafting a law, the craftsman adapting his tool. Design is the historical core of the engineer's trade — even if the contemporary engineer spends most of his time between meetings and spreadsheets, his first trade, fundamentally, is this one.

The most costly misreading: that design is an art. The idea is flattering and it is everywhere — in the schools, in the studios, in project discussions. Above all it is convenient, because if design is an art, then everyone is right, so no one is wrong, and there is no good or bad answer: every proposal is worth every other, and the debate is endless by construction. This relativism is not a guardrail against the arrogance of experts; it is a machine for endless discussion. The planning theorists Horst Rittel and Melvin Webber described this property of design problems as early as 1973, filing them among the *wicked problems*: these problems have **no stopping rule**[3]. Nothing in the nature of a design problem signals that the discussion is over or that the solution has been reached — you can always reformulate, always object, always prefer something else. The practical conclusion is

immediate: since the stopping rule will not come from the problem, it must come from the method. Validation criteria stated before the solution are what make it possible, one day, to say: this is done, and here is why. A design of which no one can say why it is good — against which criteria, under which constraints, in the service of which goal — is not a design: it is a preference that found a budget. This is also what makes the flattering formula that "everyone is a designer" false: anyone can cobble together a nightstand to their own taste; no one improvises a tool on which thousands of strangers will depend — intuition is enough for the first, never for the second.

The symmetrical claim is just as false: design is not a science. Science describes what is; design decides what will be, under constraints, for someone. It draws on scientific knowledge — without mathematics and physics, you cannot even bound the space of possible solutions — but it does not reduce to it: no equation chooses for you between two valid architectures, and the choice commits humans that equations do not describe. Nigel Cross defended the idea that design's own ways of knowing form a **third discipline**, distinct from the sciences as from the humanities, with its own objects and methods[4]. That is this book's position: design is judged — so it is not an art in the relativist sense; it is judged *against criteria that have to be built* — so it is not a science. Design is a discipline.

There is nevertheless one thing design must borrow from the experimental sciences: falsifiability. Popper made it his demarcation criterion — a theory is scientific only if it states the conditions under which it would be recognized as false[5].

Transpose it: a design is a design only if it states the conditions under which it will have failed — its criteria, stated before, verifiable after. Not to turn design into a science, but for two reasons that have nothing academic about them.

The first: the designed object will serve a public far larger than the test panel — thousands of strangers whom neither the designer's intuition nor the testers' subjectivity represents. An unfalsifiable proposal can be judged only by those who designed it; a falsifiable proposal can be judged by anyone, including those who will come after.

The second: around a single object press individual subjectivities by the thousands, and their sum will have to be resolved somewhere. There are only two ways to resolve it: by force — seniority, veto, the balance of power, and the part "Freedom and its conditions" will show what that produces — or by a rational and democratic procedure. But the democracy of design does not sit where people usually put it. You do not vote on solutions: the chapter "Listening without obeying" will say why the user has no authority over them, and the chapter "The committee will invent nothing" what committees produce. It sits on the *criteria* and the *priorities*: what will count as "better," for whom, in what order — that is, exactly where values live, and where no expertise confers authority. Criteria deliberated democratically, solutions decided against the criteria: each level gets the regime it is due. And with this we also hold what is political about falsifiability: an unfalsifiable design can be discussed only by its authors; a falsifiable design gives a handle to everyone it concerns — no

need to know how to read code to read a criterion and check that it is met. The last part of this book will build on this base.

The role of the discipline is therefore to prepare the change of scale. The statistical sciences know, in certain cases, how to estimate the uncertainty involved in using statistical results measured on a small sample of individuals to infer the general behavior of a larger population. The reasoning is mathematical, but its intuition can be explained without calculation: since we do not know to what extent our test sample faithfully represents the general population (perhaps it was not chosen entirely at random, perhaps it is the result of a filtering that nobody saw: it is therefore biased in favor of one or another of the subgroups contained in the general population), we do not know either to what extent the knowledge extracted from that sample (our statistics) can be carried over to the general population to describe it faithfully. The change of scale brings an error that cannot be measured, but whose probable value (or probable range) statistics can estimate at a certain confidence level (that is what uncertainty means). The discipline not only lets you prepare for this unavoidable error, but also allows feedback (a correction) from the general population once the proposed solution is made available, measured against the goals of the desired solution.

The serious objection to falsifiability comes from Donald Schön. Watching architects, engineers, and therapists at work, Schön showed that the experienced practitioner mobilizes a knowledge he does not know how to state — a "reflection-in-action" that overflows any explicitable technical rationality[6]. The objection is well founded: part of design knowledge is tacit, and anyone who

has designed anything knows it. But we must see what the objection establishes, and what it does not. That knowledge is tacit does not make it arbitrary. I trained as a pianist and I have taught music: the interpretation is free, the wrong note is not. The teacher who corrects a fingering mobilizes a largely tacit knowledge — he cannot always say *why* that fingering is bad — but his correction is not an opinion: it is verified in the tempo held, in the phrase that carries or breaks. The tacit is a mode of *possessing* knowledge, not an exemption from *validation*. In design, it is the same: the expert's intuition proposes, the criteria dispose. Schön describes how designers know; he provides no reason to stop holding them to account.

Designing is a discipline for solving a human problem, of which we have chosen to make a technical reading. Every word in it matters, and the coming chapters unfold it: *a discipline* — criteria stated before the solution, which give discussions their stopping rule (chapters "Stating the problem" and "Measuring without lying to yourself"); *a problem* — without a problem, you are not designing, you are tinkering (chapter "There is no technical problem"); *human* — the problem belongs to someone, and that someone is not the solution (chapter "Listening without obeying"); *a chosen technical reading* — the reading is a decision, not a fate, and other readings exist. Of these 4 terms, *human* is the one that commits us furthest, and it will take the whole book to reach the end of what it implies: if the problem belongs to someone, then designing is what we owe that someone. The word being fixed, we can turn to the thing.

1. Technical French maintains the distinction: *conception* for the engineering process, *design* for its graphic side. English covers both with *design* — which explains many a misunderstanding in the bilingual communities of software. ↩

2. Simon, H. A. (1969). *The Sciences of the Artificial*. Cambridge, MA: MIT Press (3rd ed. 1996) https://mitpress.mit.edu/9780262537537/the-sciences-of-the-artificial/ . "Everyone designs who devises courses of action aimed at changing existing situations into preferred ones." ↩

3. Rittel, H. W. J. & Webber, M. M. (1973). "Dilemmas in a General Theory of Planning". *Policy Sciences*, 4(2), 155-169 https://doi.org/10.1007/BF01405730 . Of the 10 properties they give the *wicked problems*, the second is the decisive one here: "Wicked problems have no stopping rule." ↩

4. Cross, N. (1982). "Designerly Ways of Knowing". *Design Studies*, 3(4), 221-227 DOI . ↩

5. Popper, K. R. (1934). *Logik der Forschung*. Vienna: Springer https://search.worldcat.org/title/logik-der-forschung/oclc/263117171 (French trans. *La Logique de la découverte scientifique*, Paris: Payot, 1973) — falsifiability as the demarcation criterion between science and non-science. The transposition to design is ours: it borrows the demarcation criterion, not the method. ↩

6. Schön, D. A. (1983). *The Reflective Practitioner: How Professionals Think in Action*. New York: Basic Books

https://books.google.com/books/about/The_Reflective_Practitioner.html?id=E85qAAAAMAAJ. ↩

2. There is no technical problem

Engineering schools teach that design consists in "providing a technical solution to a technical problem." The formula has the merit of symmetry and the flaw of being half false. The true half gets violated every day; the false half costs everyone dearly, starting with the user.

The true half: without a problem, you design nothing. There is no essay without a thesis question, and there is no design without a problem — that is the very definition of the previous chapter: existing situations, preferred situations, and the path from one to the other. Remove the problem, and only the path remains, which is to say the activity for its own sake. Activity for its own sake has a name: artistic creation. But art produces a unique object with no use, whereas design produces a utilitarian, repeatable object — sometimes a working tool people will come to depend on — that must serve more than a single user. Design carried out with the approach of artistic creation is called tinkering. There is a time and a place for tinkering: when you work for yourself. It becomes a design vice when it masquerades as development: when features nobody asked for pile up because they amused their author, shipped subtly broken because nothing defined what "done" meant. Programmer folklore has a formula for this drift, attributed to one Louis Srygley, of whom no trace has ever been found: "Without requirements or design, programming is the art of adding bugs to an empty text file."[1] There is also an

economic argument, more prosaic: designing costs — time, money, attention. A cost without a problem to solve is waste by definition; the problem is not an entry formality, it is what justifies the expense.

The false half lies in the adjective: there is no such thing as a *technical* problem. A problem is a gap between a current state of the world and a preferred state — and the preference always belongs to someone. A pile of raw photographs is a problem for no one; it becomes one for the photographer who must deliver 30 clean images by Monday. The problem is hers, not the machine's: the machine has no preferred states. What we call a "technical problem" is a human problem already translated — one we have chosen to read technically — and which became technical *through that reading*, not by its intrinsic nature. The nuance sounds scholastic; it is operational.

The classic parable says it in 3 lines. A customer walks in and asks for a drill. The zealous salesman compares drills; the designer asks: to drill what? — A hole, to hang a picture frame. The problem was never the drill, nor even the hole: it is hanging the frame on the wall. And for the problem "hang a picture frame," the solution space is entirely different — double-sided tape, a picture rail, and no tool to buy[2]. The lesson is not that the customer is a fool; it is that the request always arrives already translated, and that the requester's translation — he thought "drill" because he knows drills — is almost never the best one. The first act of design is to walk the translation back up to the problem, then choose your own reading.

Yet the possible readings are more numerous than the engineer spontaneously admits. Not every software problem is solved by writing code, for instance. And not every solution necessarily lives *in* the software either. When the same question comes back every week on a program's forums, you can answer it with a feature — or with a documentation page, a 3-minute video, a reorganization of the interface that makes the question moot. When insufficiently trained users flood a project with requests that betray a misunderstanding of the domain, the root of the problem is not in the code: it is addressed through teaching. The choice of reading is the first design choice, and the most structuring one, because everything else follows from it — including the bill.

Let me give a lived example, because it shows the full mechanics. An architectural assistant had spent a week fighting Visual Basic to make Excel display lists of furniture extracted from CAD software, for a new courthouse to be built. This being public procurement on a large building, the furniture invoice had to go through the budget-approval process. His reading of the problem: "make my Excel macro work." One hour of questions was enough to walk back up to the real problem — publishing readable furniture sheets from the project's data — and two hours to solve it through another reading: a script converting the CAD software's CSV export into HTML pages, plus 6 hours of graphic dressing to make it pretty. A week of dead end against a day of work: the gap does not measure a difference in skill, it measures the cost of a bad translation of the need. He was fighting with the tool he knew; the proverb says that when you hold a hammer,

everything ends up looking like a nail[3] — and it needs completing: the man who has only a hammer *translates* all his problems into nails, and it is the translation, not the hammer, that does the damage.

For the bad reading has a cost of its own, beyond the wasted effort: it produces code where none was needed. Software is not an asset that accumulates, it is a liability that must be maintained — every line written will have to be understood, tested, fixed and carried by someone, for years (the chapter "Code is a liability" does the accounting). Answering a documentation problem with code converts a one-time cost into perpetual debt. The reverse is not true: a documentation page does not break the build, does not crash the software, and gets fixed in 5 minutes. Between equivalent readings, the one that produces the least code wins.

There remains the technical purist's objection, which must be taken seriously because it still structures the training of engineers: engineering deals with machines, and humans will adapt — they have always ended up adapting. The objection has a piece of history going for it: yes, humans adapt, it is even their main talent. But it fails 3 times. First among its own: the sharpest refutation came from the very heart of industrial design, from Don Norman — what we call "human error" is almost always a design error, and blaming the user is the surest way to never fix the system[4]. Next by the costs: adaptation is not free; it is paid by someone, in learning time, in errors, in fatigue — a real cost, displaced out of the project's budget onto people who never signed up to carry it. Finally by incoherence: even the criteria the purist holds to be purely technical — performance, robustness —

only make sense relative to human ends. Performance is time given back to someone; robustness is trust someone can place in the tool. Unplug the human, and there is nothing left to optimize. And that human has a property this book will keep running into: he did not choose the machine, it fell on him, and it distanced him from the material he works with, turning the knack that allowed contextual adjustment into frozen, standardized procedures.

The problem is human, its translation is a choice, and that choice can be remade, unmade, questioned. But if the problem belongs to someone, you have to go get it from them — and that is where the trouble starts, for the one who lives the problem is also the worst placed to state it. Listening without obeying: that is the subject of the next chapter.

---

1. "Without requirements or design, programming is the art of adding bugs to an empty text file." A quotation ubiquitous in professional folklore, systematically attributed to Louis Srygley — for whom no primary source nor any biographical trace could be established. It is quoted here for what it is: anonymous wisdom of the trade. ↩

2. The parable descends from an American advertising slogan of the 1940s: "Last year over one million quarter-inch drills were sold—not because people wanted quarter-inch drills but because they wanted quarter-inch holes." Attributed to Leo McGivena (1947), popularized by Theodore Levitt, who quoted it crediting McGivena (*The Marketing Mode*, 1969) https://openlibrary.org/books/OL4433438M/The_marketing_

mode, then massively misattributed to Levitt himself. Clayton Christensen drew from it the modern theory of "jobs to be done": you don't buy a product, you hire it to do a job ("Marketing Malpractice," *Harvard Business Review*, 2005) https://hbr.org/2005/12/marketing-malpractice-the-cause-and-the-cure. ↩

3. The "law of the instrument," formulated successively by Abraham Kaplan (*The Conduct of Inquiry*, 1964) https://doi.org/10.4324/9781315131467 and Abraham Maslow (*The Psychology of Science*, 1966) https://openlibrary.org/books/OL41390694M/The_psychology_of_science: "I suppose it is tempting, if the only tool you have is a hammer, to treat everything as if it were a nail." ↩

4. Norman, D. A. (2013). *The Design of Everyday Things*, revised and expanded edition. New York: Basic Books, chap. 2 https://www.hachettebookgroup.com/titles/don-norman/the-design-of-everyday-things/9780465050659/. Norman also describes users' self-blame in the face of bad interfaces — they blame themselves for design faults — and coins for this mechanism the term *taught helplessness*. We will return to it at length in the chapter "What insularity produces". ↩

3. Listening without obeying

The previous chapter left the designer facing a difficulty: the problem belongs to someone, so you have to go get it from them — but the one who lives the problem is the worst placed to state it. Two symmetrical errors lie in wait here. The first is to obey: do what the user asks, the way he asks it. The second is to stop listening altogether: decide in his place, from the height of expertise. The first produces misshapen software; the second, software for no one. The middle way holds in 3 words: listen without obeying.

Let us be clear about what the user *can* say. A user never comes to state his need; he comes to state what he wants, and the two almost never coincide — sometimes the need is even the opposite of the request. This is not a disability, it is a consequence of his position: to imagine a solution, you must know the solution space, and the user knows only one point of it — the tool in his hands, which he will be tempted to tweak marginally for all his needs rather than choose another one, better suited to the new task he has to accomplish (see the previous chapter). Henry Ford is credited with the famous line — "If I had asked people what they wanted, they would have said faster horses"; he almost certainly never said it, the first known attribution dating from half a century after his death[1], but the apocryphal thrived because it condenses an accurate observation: the request is always phrased in the vocabulary of known solutions, and will

almost always be a minor variant of what already exists (the competitor's feature, the relocated button, the faster horse). Executing these requests one by one means patching the existing thing into Frankenstein's creature — a piece of software made of grafts, none of which answers the problem because nobody stated it. A romanticized view might see in this a tool "transcending its original function," escaping its designers' goals as its users appropriate it. That can be true to a degree, but you have to ask whether adding can-opener features to a screwdriver isn't just making a screwdriver more dangerous to use, less practical to hold, less solid for applying a specified torque — and therefore, in the end, worse at its original task. If we never stopped selling screwdrivers, hex keys and the whole panoply after inventing the Swiss Army knife or the Leatherman multi-tool, it is precisely because these polymorphic tools are handy for the occasional fix but inefficient for intensive, demanding use.

The psychology of quality has characterized why the user has such trouble defining his own need. Noriaki Kano distinguished 3 classes of expectations[2]: *implicit* expectations, so obvious to the user that he will never think to state them — their absence infuriates, their presence goes unnoticed; *expressed* expectations, on which he can take a position; and *latent* expectations, which he cannot formulate because he does not imagine they are possible. Direct questioning harvests only the middle band. The base and the summit — what goes without saying and what exceeds imagination — are invisible to the questionnaire, and they are precisely what makes tools safe and tools remarkable.

To this first limit a second one adds itself, more insidious: when the discussion leaves the real user for "the average user," it leaves reality altogether. The US Air Force learned it in 1952: out of 4,063 airmen measured, the number of those close to average on the cockpit's 10 dimensions was exactly zero[3]. The average man is a statistical fiction; designing for him is designing for no one — the cockpit solution was the adjustable, not the average. The "average user" of project discussions is worse still than that fiction: measured by nothing, he becomes elastic — he takes the shape of the argument he must serve, and everyone invokes his own[4]. The countermeasure is known: real, named, situated cases — this photographer, this volume, this deadline — and, when generalizing is needed, archetypes built from real cases, never averages.

A guardrail before leaving the cockpit, because the lesson reads too fast. "Adjustable" is not a magic word: every adjustment is a degree of freedom, every degree of freedom a moving part, and every moving part a source of failure, a source of wear, a manufacturing and maintenance cost, and one more test case — in software, every option multiplies the combinations to verify, and the chapter "In praise of the simple" will put that particular inflation on trial. The cockpit seat did not adjust everything; it adjusted the dimensions where bodies vary in a *measured* way, and where that variation prevents the task — reaching the controls. That is the complete criterion, and it has two faces: an adjustment has a price, and it is bought only where variation among real users is demonstrated and blocks the work; everywhere else, the right answer remains a fixed value, well

chosen once. Adaptability is a budget, not a virtue, and this chapter is not pleading for more knobs.

What is left to listen to, then? A lot — but not declarations. The first rule of usability, according to Jakob Nielsen: don't listen to what users *say*, watch what they *do*[5]. Declarations are reconstructed, complacent, predictive — and the predictions people make about their own behavior are only weakly correlated with their acts: stated preference predicts only a quarter of measured performance. The serious act of listening has been codified by contextual inquiry: go to the place of real work, and put yourself there in the position of an *apprentice* — the user is the master, not of the software, but of his trade, and you learn his trade the way an apprentice learns: by watching him work, by asking why, on concrete, present cases[6]. And when you must question anyway, Ulwick's rule bounds the question: ask for the expected *outcomes* — what, in the end, must be true? — never the solutions; products built on customer-specified solutions fail with remarkable regularity[7].

All of this condenses into an operational criterion, which will serve throughout this book: **the user has authority over his goals, his context and his pains — never over solutions.** Admissible is his testimony: what he does, how he does it, what it costs him — the concrete, in the present or the past. Inadmissible is his prescription: what should be done, what he would surely use, what "people" want — the normative, in the future, in the conditional. That leaves opinion — the judgment passed on a proposed or delivered solution. Before he has tried it, that opinion is just one more prediction, and falls with the others;

after a real trial, long enough to get past the discomfort of a changed habit, it has become testimony: it enters the first category, and counts.

Two lived situations show the criterion at work. A documentation maintainer — who knew the software better than I did, years of manual behind him — reproached my interface for requiring, on certain operations, more clicks than the old one, which performed many dangerous tasks on a mere hover over the element to edit. Admissible testimony: the pain is real, counted, situated. Inadmissible prescription: "put it back the way it was." For behind the click count hid other needs, mute in Kano's sense — never destroy data without warning, through one unlucky hover; stay transparent, for whoever arrives, about the target of read/write operations — and the arbitration between these needs is design work, not a vote (the chapter "Measuring without lying to yourself" does the accounting). Likewise the day I grouped the masking controls in a single place — masks are the zones that restrict an adjustment to part of the image — instead of repeating them in every module that uses them. The protest was general, and its testimony accurate: the tool is farther away than before, takes more clicks, and you have to cross the interface to reach it. But obeying — putting the controls back in every module — would have lengthened every module, on screens where telemetry showed that one user in 3 already lacks the screen height to display a single one in full. The testimony stated the cost of the change; it said nothing of the cost of the alternative, which only the designer saw. Listen to the pain, refuse the prescription: both moves, together, every time.

There remains the most serious objection, and it is twofold. The Scandinavian school of *participatory design* holds that users must co-design, for two reasons of different natures[8]. A political reason: technology restructures work, and those whose work is restructured have a democratic right to weigh on it — this is not an efficiency argument, and no usability study can refute it. An epistemic reason: the know-how of the trades is tacit, it escapes interviews as it escapes observation, and only devices of *design-by-doing* — mock-ups, prototypes to manipulate — make it speak. The epistemic objection, on examination, confirms the criterion more than it overturns it: participatory design never has the user *verbally specify* a solution — it changes the medium so that tacit competence can express itself, which is to say it enriches the testimony; mock-up work is contextual inquiry by other means. As for the political objection, it is well founded, and this book will answer it on its own ground: the last part defends co-ownership of the *goals* and the budgets by those who use the tool — deciding together *what* deserves solving is a right; the *how* remains a trade. A final word on lead users, who are known to genuinely innovate in some domains: they do not contradict the criterion, they change its subject — whoever designs his own tool has become a designer, with the duties of the post.

A division of labor, however, has a reciprocity clause, which must be stated before closing this chapter, for forgetting it overturns everything. If the designer must listen to the user without obeying him, then his own testimony — on what a solution costs, on what a change breaks, on what is feasible — must in turn be heard by the collective. Failing which, the formula inverts into

obeying without being heard. The designer is no longer the authority on the solution space; he becomes the executor of a space that others decide, ordered to repair its damage with no say on the causes. The pure form of this reversal was addressed to me one August evening in 2022, when an overhaul of the input system had once again broken a component I had fixed 3 years earlier: to my objections, a contributor replied, in 3 messages — "find the bug", "fix it", "stop fucking bitching"[9]. The sentence is a perfect theoretical object. It denies the designer the only register this chapter grants him without reserve — testimony on technical reality — while imposing on him the only thing the chapter expects of him: execution. It is "listening without obeying" turned inside out like a glove, against the one who does the work. The study of organizations knows this regime under the name of *organizational silence*: a climate where disagreement voiced upward is systematically discouraged, until those who know stop speaking and leave[10]. The complete mechanism — what happens to the one whose internal voice is stifled, and why he ends up leaving rather than shutting up — belongs to the last part of this book; it is enough here to retain that the proposed division of labor is not a disguised hierarchy, but a contract with two mutual obligations, and that either obligation left unfulfilled degenerates design into domination, regardless of who exercises it and who suffers it. And note the reach of this statement, which spills far beyond the case at hand: any relationship where one party can and the other needs tips into domination the moment it stops being accountable — true of the service rendered to the user as much as of the work demanded from the one who produces.

Listening without obeying is therefore not a posture of distrust, it is a division of labor. To the user the problem, its flesh, its context; to the designer the translation and the solution space; to honest use the verdict — and to each the right to be heard in his register. Provided the complaint is not treated where it shows itself. What the user reports is a symptom, not a cause, and the first task is to trace it back to one before acting. That is the subject of the next chapter.

---

1. No trace of the line in Ford's writings nor among his contemporaries; first known attribution in 1999, 52 years after his death (Quote Investigator, "Faster Horse," 2011 https://quoteinvestigator.com/2011/07/28/ford-faster-horse/ ; Vlaskovits, P., "Henry Ford, Innovation, and That 'Faster Horse' Quote," *Harvard Business Review*, 2011 https://hbr.org/2011/08/henry-ford-never-said-the-fast). ↩

2. Kano, N., Seraku, N., Takahashi, F. & Tsuji, S. (1984). "Attractive Quality and Must-Be Quality." *Hinshitsu — Journal of the Japanese Society for Quality Control*, 14(2), 39-48. https://doi.org/10.20684/quality.14.2_147 The 3 original classes: *must-be*, *one-dimensional*, *attractive*. ↩

3. Daniels, G. S. (1952). *The "Average Man"?* Technical Note WCRD 53-7, Aero Medical Laboratory, Wright-Patterson Air Force Base. https://apps.dtic.mil/sti/citations/tr/AD0010203 Out of 4,063 subjects, not one fell within the middle third on all 10 dimensions at once. See also Rose, T. (2016). *The End of*

Average. New York: HarperOne. https://search.worldcat.org/title/926105728 ↩

4. The "elastic user" is Alan Cooper's: *The Inmates Are Running the Asylum*. Indianapolis: Sams, 1999 https://search.worldcat.org/title/41237183 — along with his countermeasure, *personas*: precise archetypes built on real users. ↩

5. Nielsen, J. (2001). "First Rule of Usability? Don't Listen to Users." Nielsen Norman Group. https://www.nngroup.com/articles/first-rule-of-usability-dont-listen-to-users/ The correlation between stated preference and measured performance, about 0.44, comes from Nielsen, J. & Levy, J. (1994). "Measuring Usability: Preference vs. Performance." *Communications of the ACM*, 37(4). https://doi.org/10.1145/175276.175282 ↩

6. Beyer, H. & Holtzblatt, K. (1997). *Contextual Design: Defining Customer-Centered Systems*. San Francisco: Morgan Kaufmann. https://search.worldcat.org/title/37443490 The interview model there is explicitly that of master and apprentice. ↩

7. Ulwick, A. W. (2002). "Turn Customer Input into Innovation." *Harvard Business Review*, January 2002. https://hbr.org/2002/01/turn-customer-input-into-innovation ↩

8. Ehn, P. (1988). *Work-Oriented Design of Computer Artifacts*. Stockholm: Arbetslivscentrum

https://books.google.com/books/about/Work_oriented_Design_of_Computer_Artifac.html?id=7dJQAAAAMAAJ; Sundblad, Y. (2011). "UTOPIA: Participatory Design from Scandinavia to the World," in *History of Nordic Computing 3*. Berlin: Springer. https://doi.org/10.1007/978-3-642-23315-9_20 On the lineage with lead-user innovation: Björgvinsson, E., Ehn, P. & Hillgren, P.-A. (2010). "Participatory Design and 'Democratizing Innovation'." *Proceedings of the 11th Biennial Participatory Design Conference*, ACM. https://doi.org/10.1145/1900441.1900448 ↩

9. Log of the project's public Matrix chat room (bridge of the IRC channel), export of July 23, 2026 in my possession. On August 8, 2022 at 10:06 p.m., in 3 messages: "find the bug", "fix it", "stop fucking bitching". The sequence this exhibit establishes is the one that matters here: a component I had fixed in 2019; broken by an overhaul I had not been able to review; mine to carry again — find it, fix it, and shut up — with no help offered. The full context of that evening is filed in the record of the chapter "The closed circle". ↩

10. Morrison, E. W. & Milliken, F. J. (2000). "Organizational Silence: A Barrier to Change and Development in a Pluralistic World." *Academy of Management Review*, 25(4), 706-725. https://doi.org/10.2307/259200 See also Van Dyne, L., Ang, S. & Botero, I. C. (2003). "Conceptualizing Employee Silence and Employee Voice as Multidimensional Constructs." *Journal of Management Studies*, 40(6), 1359-1392. https://doi.org/10.1111/1467-6486.00384 ↩

Part II — The discipline

4. The root and the symptom

A problem shows up where it hurts, not where it is. Physicians know this — referred pain radiates far from its lesion — engineers forget it: the place where a defect manifests is a clue, not an address. The previous chapter taught how to collect the pains; this one teaches not to treat them on the spot. It is a delicate art, because everything — urgency, economics, psychology — pushes us to act where we can see and to ignore the rest.

The stakes can be counted. A root problem carries branches: treated at the root, it resolves them all at once, and the solution is usually simpler and more general than the sum of the patches its manifestations would have required. Treated at the branches, it grows back — and every patch adds surface to the system: more code to understand, more interactions to break. Every maintainer knows the extreme form of this drift: whack-a-mole, where each fix makes a defect pop up somewhere else, until the software no longer *contains* bugs — it *is* one. At that stage, no local treatment makes sense anymore; it is the sign that the symptoms have been treated long enough for the disease to invade everything. Symptomatic treatment sometimes has an elegance that betrays it: watching a program "fix" its slow startup by displaying a splash screen while it loads is watching a symptom dressed up as

a feature — the loading time has not dropped by a millisecond, someone just repainted the waiting room.

If symptomatic treatment is so bad, why is it so universal? Because it is rational — locally, in the short term, for each individual. The philosopher Abraham Kaplan called this the principle of the drunkard's search: the man looking for his keys under the streetlight, not because he lost them there, but because, as he explains, "it's lighter here!"[1] This is known as the streetlight effect, or streetlight bias. We act where we can see, because seeing makes action possible and cheap, and because acting gives us a feeling of control over the situation. Notice that sitting on your hands until you understand the problem well enough to be reasonably sure you actually hold its cause is rarely an audible option, when it doesn't get dismissed outright as a perfect-solution fallacy. Yet the point here is not to oppose a symptomatic solution, temporarily and locally better, to sheer inaction on the grounds that the symptomatic solution would not be perfect: the point is to do nothing that could damage the overall system in the long run until you have finished mapping the extent of the problem and its origin — otherwise the symptomatic solutions will have to multiply across the different floors of the system. System dynamics formalized what comes next: in the archetype Peter Senge names *Shifting the Burden*, the symptomatic solution relieves faster and cheaper than the fundamental solution — and each relief reduces the pressure to undertake the fundamental solution, whose capacity *atrophies*; the organization becomes dependent on its patch as on a painkiller, in increasing doses[2]. Nobody was stupid at any point:

everyone followed the slope of the maximal (or, at least, maximally visible) result for minimal effort, all of it seen in the short term. That is precisely why a discipline is needed — you don't climb back up a slope by inclination.

The discipline exists, and it comes from factories. At Toyota, Taiichi Ohno had reduced it to a one-line instruction: facing a problem, ask *why* 5 times. A machine stops. Why? — The fuse blew, overload. Why? — The bearing was poorly lubricated. Why? — The lubrication pump wasn't delivering properly. Why? — Its shaft was worn. Why? — With no strainer, metal shavings were getting into the pump. Replacing the fuse restarts the machine today and guarantees the same breakdown in 3 months; fitting a strainer makes it disappear for good[3]. The fuse is where the problem shows; the strainer is where it is decided. Between the two, 5 questions — and the art lies less in the counting than in the crossing: each serious *why* changes floors, from the component to the subsystem, from the subsystem to the architecture, from the architecture to the process that produced the architecture. The same movement holds for needs, which are Russian dolls: if two needs still seem to contradict each other, you are usually holding branches, not the root — you haven't climbed high enough.

Industrial safety gave this verticality its definitive names. James Reason distinguishes *active* failures — the end-of-chain error, visible, immediate, attributable to someone — from *latent* conditions: "resident pathogens" installed in the system long before, by decisions of design, organization, budget, and waiting for their opportunity[4]. The accident is almost never the active

error alone; it is the encounter of the active error with latent conditions nobody was watching, because they hurt nowhere. Where the problem is seen: at the end of the chain. Where it comes from: upstream, years earlier.

For a few years, I was the end of the chain. The tone-mapping module I had written — *filmic*, which compresses the scene's dynamic range into the screen's — sat, by construction, at the end of the pipeline: it received whatever every previous module had produced, and made it visible. It brightened the shadows, so it revealed the sensor noise that crushed shadows used to hide; it preserved saturation, so it rendered as-is the clipped highlights that upstream had let degrade to magenta, where the old renderings had discreetly washed them out; and it inherited, garbage in, garbage out, every flaw of what came before it. So every complaint came to me — noise, magenta, wrong colors —, addressed to the module that showed them, never to the ones that manufactured them: one forum thread asked why filmic "render[s] clipped areas so weirdly"[5], when clipping is a fact of the sensor, handled — or not — well upstream. That is blaming the thermometer for the fever. The mechanism is documented: ordinary causal judgment singles out the *abnormal* condition — what just changed, what is easiest to imagine absent — and leaves out of frame the background conditions that did the work[6]; in a chain, it credits the last link that made the visible difference[7]; and we hold it against the bearer of bad news, even one we know had nothing to do with it[8]. Industrial safety has a word for this: blaming the *sharp end* — the operator, the visible component — for latent conditions manufactured at the *blunt end*[9]. And

debugging has measured the distance: in a census of their worst bugs, professional programmers attribute more than half of the difficulty to the temporal and spatial *chasm* between cause and symptom[10]. The module at the end of the chain will therefore receive the complaints of the whole pipeline; expect it, and do not answer by making it hide — say, by washing out the highlights — what upstream broke: the defect would remain, it would just no longer be seen.

I lived the software version of this theorem. A computation cache — the part that avoids re-running heavy computations the previous run had already done — had been dragging 10-year-old bugs: ghost images, spurious recomputations, inconsistencies fixed one by one as they showed. One of these patches — a forced refresh of the whole display by flushing the entire cache, which defeated its whole reason for existing — was added, removed because of its resource consumption, reintroduced behind an arbitrary timer, hardened after crashes, then finally deleted: 10 years of symptomatic treatment of a single symptom. The climb back up, when it finally happened, followed Ohno's floors: fixing the cache required fixing the computation pipeline; fixing the pipeline required making the data's life cycle visible — what is valid, what is stale, who decides. The cache bugs were not bugs *in* the cache: they were the symptoms of an engine whose responsibilities were tangled. Once the life cycle was made explicit, the fixes stopped being patches — each one *removed* code instead of adding it. I drew from it a practical rule, more effective than all the exhortations to "look for the root cause": **refuse any fix that does not simplify the code.** A patch adds — a special

case, a lock, a timer; a root treatment subtracts, because it removes the misunderstanding that made those defenses necessary. Facing a proposed fix, it is enough to look at what it does to the code: if it adds some, you probably haven't reached the root yet. And if it does add code, you should still ask whether it is there to complicate the existing logic and its heuristics, or on the contrary to add diagnostic tools, safety nets and exception detection *upstream*. The number of lines of code is one indicator among others, not the metric to optimize. The metric to optimize, here, is the terminal complexity of the system, which is settled by handling anomalies once, as early as possible at the program's inputs, so as to avoid armoring every possible output with fallbacks for all the error and exception cases the upstream let through.

The hunt for the root has, however, an opposite excess, which must be prevented: looking for a cause where there is none. Every system fluctuates — a production line never outputs two rigorously identical parts, a program never executes the same task twice in exactly the same time. W. Edwards Deming called that variation *common causes*: the system's background noise, attributable to no particular event; and he reserved the name *special causes* for the deviations behind which there really is an identifiable event — Ohno's breakdown is one. The distinction dictates the course of action, because correcting one by one deviations that are only noise stabilizes nothing: it makes things worse. Deming demonstrated it with the funnel experiment: drop marbles through a funnel above a target, and "correct" the funnel's position after each fallen marble, according to the gap

between its landing point and the target, hoping to correct the trajectory by successive refinements. Every correction rule tried scatters the impacts more than touching nothing does: on average, the uncorrected system was already at its optimum of stability; correcting after each individual event only made it more unstable[11]. This is known as overcorrection.

Chasing the cause of a normal fluctuation is not rigor, it is derangement. The lesson does not contradict the 5 *whys*: it sets their condition of use. Before climbing from a symptom to its cause, make sure the symptom *has* a cause — that the deviation stands out from the system's normal noise. If it does not, it has no root of its own; only the overall variation has one, and that is reduced only by redesigning the system — or not at all. Telling the significant deviation from the noise is a problem of measurement and statistics; the chapters "Measuring without lying to yourself" and "Counting the absent" provide the instruments.

Noise is not the only false cause; coincidence is another, and it is more persuasive because it comes with a story. Logic has named it since antiquity — *post hoc ergo propter hoc*: after this, therefore because of this — and software offers it ideal ground, because every change there is dated, public and followed. Publish a change: everything that goes wrong in the following days will be pinned on it, including in parts of the program it does not touch and cannot touch. I expected training to protect against this reflex; experience showed me the opposite. The most assured diagnoses of this kind came to me from seasoned scientists and programmers, in their fifties or sixties, who followed the changes

more closely than the average user — and who arrived with a formed hypothesis, a plausible explanation, and the authority to be heard. That is where the trap closes: you take them seriously, because they have the credentials for it; you follow their lead for half a day, a day; you get nowhere; and only then do you begin to doubt the diagnosis, before finding that it was wrong from the start — plausible, authoritative, and wrong. Experimental psychology has documented every stage of this mechanism. To attribute a cause, we do not count the cases where the effect follows the presumed cause and those where it does not; we rely on *cues* — what came just before, what is nearby, what stands out. In the laboratory, subjects name as the cause the event that shortly preceded the effect, even when the observed frequencies show that another event is responsible; and the further the real cause is moved away in time from its effect, the less they believe it is the cause[12]. Skills do not immunize: mathematical psychologists gathered at a conference held the same false intuitions about small samples as the public; most reasoning biases are independent of cognitive ability; higher numeracy sometimes worsens the motivated reading of data; and expert intuition is only reliable in domains that correct it with fast, regular feedback — which a program you do not debug yourself does not provide[13]. Two things, on the other hand, do work, and they add up. Confidence: between two opinions, people follow the one delivered with the most conviction, not the one that has most often proved right. And credentials: the same message persuades more when it comes from a source deemed competent, whatever its content[14]. An error asserted with aplomb by an expert collects both premiums, and travels further than the same

error put forward by anyone else. The last stage is the one that costs: a plausible hypothesis, once stated, steers the whole investigation — clinicians handed a suggested diagnosis *perceive* different signs, the label gathers a *diagnostic momentum* that later observers no longer dislodge, and premature closure is the leading cognitive cause of medical error[15]; software testers mostly write tests that confirm, and the confirmation bias measured in developers predicts the defect density of their code[16]. The practical consequence rejoins the criterion of the chapter "Listening without obeying": the reporter has authority over the symptom — what he saw, when, how to reproduce it —, never over the cause. An unverified diagnosis is not one more piece of information, it is an anchor; those whose job is fixing bugs know it, since they rank reproduction steps and execution traces at the top of what they expect from a report, and the reporter's theory nowhere[17]. The best contribution an expert can make to a problem he has not instrumented himself is therefore the most modest one: describe, date, reproduce — and keep the hypothesis to himself until he has tested it.

At the top of the climb, there is something better than solving. Russell Ackoff distinguished 4 treatments of a problem:

- *absolve* it (ignore it and hope it goes away),
- *resolve* it (find an acceptable outcome),
- *solve* it (find the optimum),
- *dissolve* it — redesign the system or its environment so that the problem can no longer occur in it[18].

Ohno's strainer does not solve the breakdown: it dissolves it. The explicit life cycle does not fix the cache bugs: it removes their category. Dissolution is the designer's horizon: not to win against the problem, but to take away its habitat.

A whole school of complex-systems safety objects to this that the "root cause" is a category error. Sidney Dekker: the root cause is not a property of the world, it is the place where the investigation *decided to stop* — an artifact of the investigation's budget and politics. Nancy Leveson: the causal-chain model is itself an antique; accidents are failures of *control* in an entire socio-technical hierarchy, and if you remove the triggering event, another one would have served[19]. The objection is sound — and on examination, it radicalizes this chapter instead of refuting it. For what Dekker and Leveson attack is the premature end of the climb: the investigation that stops at "human error", at the fuse, at the broken component — which is exactly symptomatic treatment, moved up one floor. Their prescription — redesign the control structure rather than excise an event — is Ackoff's dissolution applied to socio-technical systems. The real disagreement is narrow: it bears on where the climb terminates — a node, or a network. The answer is to define the word rather than defend it: the root is not a metaphysical origin, it is **the deepest level at which one can still act**. In an industrial accident, that level is often a network of conditions, and the singular is a convenient fiction. In a designed artifact — a program, a machine — the regression usually ends at a decision someone made, and which can be made again: the architecture is the bottom of the well. That is the designer's privilege over the

accident investigator: his system has a blueprint, and the blueprint has an author.

The chapter's route fits in one line: climb from the symptom to the decision that produces it, learn to recognize along the way the noise that has no cause and the coincidence that is not one, and aim for dissolution rather than victory. What was found at the bottom of the well must now be written down — turned into a stated problem, with its constraints and its criteria. That is the subject of the next chapter.

---

1. Kaplan, A. (1964). *The Conduct of Inquiry: Methodology for Behavioral Science*. San Francisco: Chandler https://books.google.com/books/about/The_Conduct_of_Inquiry.html?id=_Kq_TXjfkj4C — the "principle of the drunkard's search". The same book states the law of the instrument met in the chapter "There is no technical problem": one book, two infirmities of inquiry. ↩

2. Senge, P. M. (1990). *The Fifth Discipline: The Art and Practice of the Learning Organization*. New York: Doubleday/Currency https://www.worldcat.org/title/fifth-discipline-the-art-and-practice-of-the-learning-organization/oclc/21226996 — the archetypes *Shifting the Burden* (the symptomatic solution atrophies the fundamental one) and *Fixes that Fail* (the short-termist fix worsens the problem it treats). ↩

3. Ohno, T. (1988). *Toyota Production System: Beyond Large-Scale Production*. Cambridge, MA: Productivity Press https://doi.org/10.4324/9780429273018 (Japanese ed. 1978) — the section "Repeating Why Five Times" gives the machine-stop example; the book's glossary, under "Five Why's": "By repeating why five times, the nature of the problem as well as its solution becomes clear." ↩

4. Reason, J. (1990). *Human Error*. Cambridge: Cambridge University Press https://doi.org/10.1017/cbo9781139062367, notably ch. 7 ("Latent errors and systems disasters") — active failures versus latent conditions, the "resident pathogens"; origin of the so-called Swiss-cheese model. ↩

5. Thread "Why do CAT and filmic render clipped areas so weirdly", pixls.us, 2021 https://discuss.pixls.us/t/solved-why-do-cat-and-filmic-render-clipped-areas-so-weirdly/26426 — the title attributes to the tone-mapping module a clipping produced by the sensor and inherited from the whole chain; the same thread is filed, for other reasons, in the chapter "The well-tempered conflict". ↩

6. Hilton, D. J. & Slugoski, B. R. (1986). "Knowledge-based causal attribution: The abnormal conditions focus model". *Psychological Review*, 93(1), 75-88 https://doi.org/10.1037/0033-295X.93.1.75 — ordinary causal attribution isolates the *abnormal* condition against a background of conditions taken as normal, in the lineage of Hart, H. L. A. & Honoré, T. (1985). *Causation in the Law*, 2nd ed. Oxford: Clarendon Press. Same mechanism through

counterfactuals: Kahneman, D. & Miller, D. T. (1986). "Norm Theory: Comparing Reality to Its Alternatives". *Psychological Review*, 93(2), 136-153 https://doi.org/10.1037/0033-295X.93.2.136 — the event easiest to "undo" in thought receives the blame. ↩

7. Spellman, B. A. (1997). "Crediting Causality". *Journal of Experimental Psychology: General*, 126(4), 323-348 https://doi.org/10.1037/0096-3445.126.4.323 — in a chain of causes, each is credited in proportion to how much it *changed* the probability of the outcome given what preceded it: the last link that makes the visible difference collects a disproportionate share of the blame. ↩

8. John, L. K., Blunden, H. & Liu, H. (2019). "Shooting the Messenger". *Journal of Experimental Psychology: General*, 148(4), 644-666 https://doi.org/10.1037/xge0000586 — 11 experiments: the bearer of bad news is judged less likeable even when demonstrably innocent (a random draw), the effect being driven by the need to give an agent to what goes wrong. See already Rosen, S. & Tesser, A. (1970). "On Reluctance to Communicate Undesirable Information: The MUM Effect". *Sociometry*, 33(3), 253-263 https://doi.org/10.2307/2786156 . ↩

9. Cook, R. I. & Woods, D. D. (1994). "Operating at the Sharp End: The Complexity of Human Error", in Bogner, M. S. (ed.), *Human Error in Medicine*. Hillsdale, NJ: Erlbaum, 255-310 https://psycnet.apa.org/record/1994-98177-007 — the sharp end inherits the latent conditions created at the blunt end (design, organization) and absorbs the blame for them,

because it is the visible locus of failure; the active/latent distinction is Reason's (1990), cited above. ↩

10. Eisenstadt, M. (1997). "My Hairiest Bug War Stories". *Communications of the ACM*, 40(4), 30-37 https://doi.org/10.1145/248448.248456 — 59 debugging stories from professionals: more than half of the difficulty lies in the "temporal or spatial chasm" between root cause and symptom. The general model — defect → infection → failure, the failure being only the observable end of a propagation — is in Zeller, A. (2009). *Why Programs Fail: A Guide to Systematic Debugging*, 2nd ed. Burlington, MA: Morgan Kaufmann, ch. 1. ↩

11. Deming, W. E. (1986). *Out of the Crisis*. Cambridge, MA: MIT Center for Advanced Engineering Study https://doi.org/10.7551/mitpress/11457.001.0001 — common causes and special causes (in the lineage of Shewhart, W. A., *Economic Control of Quality of Manufactured Product*, Van Nostrand, 1931 https://archive.org/details/in.ernet.dli.2015.150272); the funnel experiment, pp. 327-332. ↩

12. Einhorn, H. J. & Hogarth, R. M. (1986). "Judging Probable Cause". *Psychological Bulletin*, 99(1), 3-19 https://doi.org/10.1037/0033-2909.99.1.3 — the "cues to causality" (temporal order, contiguity, similarity) are used even when they mislead. Lagnado, D. A. & Sloman, S. A. (2006). "Time as a Guide to Cause". *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 32(3), 451-460 https://doi.org/10.1037/0278-7393.32.3.451 — temporal order

overrides covariation. Buehner, M. J. & May, J. (2003). “Rethinking Temporal Contiguity and the Judgement of Causality”. *Quarterly Journal of Experimental Psychology A*, 56(5), 865-890 https://doi.org/10.1080/02724980244000675 — causal conviction decays with delay. Review: Matute, H. *et al.* (2015). “Illusions of Causality”. *Frontiers in Psychology*, 6, 888 https://doi.org/10.3389/fpsyg.2015.00888. ↩

13. Tversky, A. & Kahneman, D. (1971). “Belief in the Law of Small Numbers”. *Psychological Bulletin*, 76(2), 105-110 https://doi.org/10.1037/h0031322; Stanovich, K. E. & West, R. F. (2008). “On the Relative Independence of Thinking Biases and Cognitive Ability”. *Journal of Personality and Social Psychology*, 94(4), 672-695 https://doi.org/10.1037/0022-3514.94.4.672; Kahan, D. M., Peters, E., Dawson, E. C. & Slovic, P. (2017). “Motivated Numeracy and Enlightened Self-Government”. *Behavioural Public Policy*, 1(1), 54-86 https://doi.org/10.1017/bpp.2016.2; Kahneman, D. & Klein, G. (2009). “Conditions for Intuitive Expertise: A Failure to Disagree”. *American Psychologist*, 64(6), 515-526 https://doi.org/10.1037/a0016755. On field experts reversing their own verdict under biasing context: Dror, I. E., Charlton, D. & Péron, A. E. (2006). “Contextual Information Renders Experts Vulnerable to Making Erroneous Identifications”. *Forensic Science International*, 156(1), 74-78 https://doi.org/10.1016/j.forsciint.2005.10.017. I stick to “experience does not immunize”: the literature on age is mixed, and does not support older decision-makers being more biased in general. ↩

14. On confidence: Price, P. C. & Stone, E. R. (2004). “Intuitive Evaluation of Likelihood Judgment Producers: Evidence for a Confidence Heuristic”. *Journal of Behavioral Decision Making*, 17(1), 39-57 https://doi.org/10.1002/bdm.460; nuance: Tenney, E. R., MacCoun, R. J., Spellman, B. A. & Hastie, R. (2007). “Calibration Trumps Confidence as a Basis for Witness Credibility”. *Psychological Science*, 18(1), 46-50 https://doi.org/10.1111/j.1467-9280.2007.01847.x — confidence lends credibility by default, until its calibration errors become visible; which is exactly what the lost half-day ends up making visible. On credentials, the founding experiment on source credibility: Hovland, C. I. & Weiss, W. (1951). “The Influence of Source Credibility on Communication Effectiveness”. *Public Opinion Quarterly*, 15(4), 635-650 https://doi.org/10.1086/266350 — the same text, attributed to a high- or a low-credibility source, changes more opinions when attributed to the former. The two axes are independent: a layman can be confident, an expert cautious. ↩

15. LeBlanc, V. R., Brooks, L. R. & Norman, G. R. (2002). “Believing Is Seeing: The Influence of a Diagnostic Hypothesis on the Interpretation of Clinical Features”. *Academic Medicine*, 77(10 Suppl.), S67-S69 https://doi.org/10.1097/00001888-200210001-00022; Croskerry, P. (2003). “The Importance of Cognitive Errors in Diagnosis and Strategies to Minimize Them”. *Academic Medicine*, 78(8), 775-780 https://doi.org/10.1097/00001888-200308000-00003 — origin of the term *diagnostic momentum*;

Graber, M. L., Franklin, N. & Gordon, R. (2005). "Diagnostic Error in Internal Medicine". *Archives of Internal Medicine*, 165(13), 1493-1499 https://doi.org/10.1001/archinte.165.13.1493 — of 100 errors, cognitive factors in 74%, premature closure first. The bridge to coincidence: Mamede, S. *et al.* (2010). "Effect of Availability Bias and Reflective Reasoning on Diagnostic Accuracy Among Internal Medicine Residents". *JAMA*, 304(11), 1198-1203 https://doi.org/10.1001/jama.2010.1276 — a recent similar case degrades the next diagnosis; deliberate reflection repairs it. ↩

16. Nickerson, R. S. (1998). "Confirmation Bias: A Ubiquitous Phenomenon in Many Guises". *Review of General Psychology*, 2(2), 175-220 https://doi.org/10.1037/1089-2680.2.2.175; Leventhal, L. M., Teasley, B. E. & Rohlman, D. S. (1994). "Analyses of Factors Related to Positive Test Bias in Software Testing". *International Journal of Human-Computer Studies*, 41(5), 717-749 https://doi.org/10.1006/ijhc.1994.1079; Çalıklı, G. & Bener, A. (2013). "Influence of Confirmation Biases of Developers on Software Quality: An Empirical Study". *Software Quality Journal*, 21(2), 377-416 https://doi.org/10.1007/s11219-012-9180-0. ↩

17. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R. & Zimmermann, T. (2008). "What Makes a Good Bug Report?". *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 308-318 https://doi.org/10.1145/1453101.1453146; extended version: Zimmermann, T. *et al.* (2010). *IEEE Transactions on Software*

Engineering, 36(5), 618-643 https://doi.org/10.1109/TSE.2010.63 — 466 developers from Apache, Eclipse and Mozilla rank reproduction steps, traces and test cases at the top of what is useful; the gap between what reporters supply and what developers expect is the very object of the study. ↩

18. Ackoff, R. L. (1981). "The Art and Science of Mess Management". *Interfaces*, 11(1), 20-26 https://doi.org/10.1287/inte.11.1.20: "There are three kinds of thing that can be done about problems — they can be *resolved, solved,* or *dissolved*"; to dissolve a problem is "to change the nature, and/or the environment, of the entity in which it is imbedded so as to remove the problem" — the problem dissolver "looks for dissolutions in the containing whole rather than solutions in the contained parts". The fourth treatment, *absolution*, and the explicit ranking of dissolution above the other three belong to his later formulations of the scheme — e.g. Ackoff, R. L., Magidson, J. & Addison, H. J. (2006). *Idealized Design: Creating an Organization's Future*. Upper Saddle River, NJ: Wharton School Publishing. ↩

19. Dekker, S. (2014, 3rd ed.). *The Field Guide to Understanding "Human Error"*. Farnham: Ashgate https://doi.org/10.1201/9781317031833 — the section "There Is No Root Cause". Leveson, N. G. (2011). *Engineering a Safer World: Systems Thinking Applied to Safety*. Cambridge, MA: MIT Press https://doi.org/10.7551/mitpress/8179.001.0001 . In the same vein: Hollnagel, E. (2014). *Safety-I and Safety-II.*

Farnham: Ashgate https://doi.org/10.1201/9781315607511 . Reason himself (1990) already held latent conditions to be plural and combinatorial — the "root cause" in the singular is a popularization of causal analysis, not its best version. ↩

5. Stating the problem

What was found at the bottom of the well does not exist yet: as long as the problem is not written down, everyone carries a different version of it, and those versions diverge in silence until the day they meet — usually in the middle of development, at the worst possible moment. Programmer folklore says it in its own way: "Without requirements or design, programming is the art of adding bugs to an empty text file"[1]. Writing the problem down is not an administrative formality; it is the act that makes it exist as a common object — something that can be passed from hand to hand, contested, amended, and above all closed. The chapter "The word and the thing" established it: design problems have no stopping rule, so the rule must come from the method. Here it is. The problem stated in writing, with its validation criteria, *is* the stopping rule.

What must be written? Everything the optimal solution depends on — and it depends on much more than the statement. For the same problem, the best solution changes with the way you look at it, the budget you have, the priorities you have set for yourself — including your values and your politics (no, that is not a dirty word, and the politicized engineer is no more dogmatic than the next one because his principles dare to show themselves in broad daylight) —, the person who will use the solution, and the lifespan you expect from that solution. Five parameters, and none of them technical: they are decisions — human, arbitrary,

debatable, situated. A stated problem is a statement plus these 5 decisions, owned in writing, as a contract of what one commits to solving. As long as they stay implicit, each participant in the discussion fills them in with his own, and we fall back into the endless debates of the chapter "The word and the thing" — not because people are in bad faith, but because they are solving different problems without knowing it, and the effort of agreeing on the definition of the object was never made.

The heart of the statement is the use case: a defined task, performed by a defined user, in a defined time. Editing a posed portrait, by a wedding photographer, between file handoff and album delivery. All 3 terms are necessary. A task without a user is an abstraction; a user without a task is a marketing target; without a time horizon, you can neither size nor validate. And if there is no use case, there is no problem — hence nothing to design, and the best move is not to open the code editor. "It would be cool if…" is not a use case; it is how collections of incoherent gadgets get made.

It remains permitted, and even necessary, to prototype without a use case — to explore an idea, learn a technique, check whether a lead deserves a specification or not. But that activity has a name, and it is not designing: it is research. The distinction is not one of vocabulary, it is one of circuit. Research produces knowledge; design produces a tool. The former has the right to fail, to stay unfinished and to end up in the trash — that is even its normal regime; the tool, for its part, will be maintained for years in the hands of people who depend on it. Disaster begins when the circuit short-circuits: when the research prototype, having failed

to find its use case, ships anyway — poured straight into the hands of end users, with its unverified assumptions and its test-tube ergonomics. Research must never reach the end user without an intermediary; between it and him stands precisely this whole book — the use case found, the problem stated, the criteria, the validation. “It would be cool if…” is an excellent start for research, and a catastrophe as a roadmap.

Then comes the most avoided decision: the scope — for whom, under what conditions. It is avoided because it excludes, and excluding is uncomfortable; but there is no choice: a tool that claimed to serve everyone would suit no one. For a trade tool, you cannot serve a professional and a beginner/hobbyist in the same way. Indeed, education and training condition perception itself. A study on visual assessment in the food industry — tasting panels judging the color of orange juice — measured the gap: after training, subjects distinguished color differences almost twice as fine as before[2]. What the trained eye sees, the fresh eye does not — literally. Designing the same tool for the beginner and for the expert is therefore not serving two audiences: it is frustrating both, because one needs to have hidden and simplified exactly what the other needs to control. What is a secondary detail for one is a priority for the other. And the trade alone does not define the scope: the photojournalist transmitting his images from the event, the fine-art photographer working slowly toward an exhibition print, and the fashion team aiming for the same quality at industrial cadence share the same trade of image-making, but do not share the same conditions of use. I ended up writing it down in black and white for my own software: it

expects from its users an intermediate to advanced knowledge of color theory and photographic technique, and does not target beginners. You can dispute that choice; you can no longer pretend it was not made. Choosing whom you design for is a values decision — and you must not try to pass it off as a statistical result. You must even look at it for what it is: choosing whom you design for is choosing whom you give up serving, in order to serve the others properly. The part "Freedom and its conditions" will measure what a socially homogeneous room settles this way without even noticing that it is settling anything. All of this is arbitrated by naming clearly, and honestly, whom the solution excludes.

There remains to write what will be binding: the validation criteria. A specification details the conditions the solution will have to meet in order to be accepted, expressed in measurable, or at least objective, quantities: the number of operations to accomplish the task, the time required, the training needed before using it, the standards it will have to interface with, and the prohibitions — the postures and gestures that injure, for the keyboard and the mouse inflict repetitive strain injuries as real as those of machine tools. Two rules govern this document. The first comes from the chapter "There is no technical problem": no technology, no device in the statement — the specification says what must be true at the end, never by what means it must become so. The second is a rule of order: the criteria are set *before* looking for solutions, because the reverse never happens — once you hold a solution you like, you write the criteria it meets. Designing is work where you have to think against your own

brain, which is lazy and falls back into its habits on its own; the strict separation between the problem and the solution is its guardrail.

The document also changes the nature of conflicts. Take a real case, lived on my own software, whose interface has two views: the lighttable, where you sort images, and the darkroom, where you edit them. Two camps of users demanded incompatible things there: some wanted the sorting tools duplicated in the darkroom, others wanted image settings duplicated in the lighttable — each camp demanding the other view's functions in its own, and each duplication lengthening an already saturated interface. No negotiation over solutions can succeed in those terms: the demands contradict each other by construction, the screen is not extensible, and functional duplication also duplicates (at least in part) the volume of code, the technical debt and the maintenance cost. Facing this dead end, the modern reflex has a respectable name, compromise, and a usurped reputation for virtue. Look at what it really is: a resolution of the conflict by sharing the losses — each party gives up half of its demands, and the arrangement passes for fair because everyone loses equally. "I agree to suffer since you suffer too": that is the morality of compromise, and it takes a strange indoctrination to see a solution in it. For the arithmetic of needs does not do half-counts: half satisfied equals, most of the time, completely frustrated. One evening, my partner wants to go to the movies and I want to go to the restaurant. The compromise would be to cut dinner short so as to catch only the second half of the film: an evening ruined for two, at full price — might as well stay home.

Logic textbooks know the general form of the error, the *argumentum ad temperantiam*: a position being halfway between two others does not make it right — the middle is just one more position, which has to prove itself like the others. In design, the midpoint between two solutions is, by definition, optimal for no one. We are back to the same problem as the "average user".

Compromise is not a solution: it is the symptom of a needs analysis interrupted too early. Negotiation theorists have shown where the unblocking happens: not between *positions* — what each side demands — but between *interests* — what each side needs[3]. Back to the evening. Why the restaurant? Because I am too tired to cook and do the dishes. Why the movies? To escape for an hour or two into an imaginary world. Nobody *needs* the restaurant or the movie theater as such; the need bears on what these places provide — and once the needs are laid bare, they no longer contradict each other: we want the same thing, to unwind. Takeout and a movie at home: both needs served in full, for less than the compromise that would have mutilated them both. The same goes for our two views, and the climb was less obvious there than for the evening. Why want to sort in the darkroom? Because switching to the lighttable cost more than a second of frozen interface, at every round trip, hundreds of times per session. Why want to edit in the lighttable? The same answer, in the other direction. Nobody needed the duplicates: the common need, which all the positions masked, was *not to lose time when switching from one view to the other*. The solution therefore lay in none of the demands, nor in splitting them: it was under the interface, in the engine — make the switch instantaneous by

avoiding recomputations, through caching (that thing again). Once that was done, each view could be specialized for its task and stripped of its duplicates: both camps served in full, by *less* interface — a single design, without variant or mode, born of the common interest the positions were hiding. Finding that object is design's job, not negotiation's. You recognize the movement of the chapter "The root and the symptom": when two requirements seem to contradict each other, you are holding branches, not yet roots — you haven't climbed high enough. Compromise gets chosen by default because that climb costs time and a skill that has nothing technical about it; but carried through, it leads to solutions that win on both fronts — and that are generally simpler, hence cheaper to maintain.

There remains the residual case: needs that, after an honest climb, remain incompatible. The answer is still not the middle — it is separation: two products, two configurations, two modes, each whole for its audience. When compromises become unavoidable in series, that is not a fate to accept, it is the sign that a single solution is trying to cover two distinct problems, and that it must be split. For stacking compromises has a precise human cost: condemning people to work only with suboptimal solutions, to invent workarounds for themselves that then get imprinted into their habits — and when they revolt against this accumulation, we diagnose, with a touch of condescension, a "resistance to change". It is perfectly healthy to fight compromises: accepting compromise means accepting failed design as a fatality, accepting to lose a bit of your comfort and your productivity with every new change. That cannot be a rational posture, in the long run.

Once the problem is stated, what follows is a marked trail, which engineering schools teach: an ideation session first — every idea written down, even the absurd ones, for they unlock associations —, then triage on feasibility, compared mockups, prototype, and confrontation of the prototype with the validation tests written in the specification; you iterate or you go to production. I will not detail the facilitation techniques for these sessions — methods exist, none is sacred. What is sacred is the order: the problem, the criteria, then only the ideas. An ideation session without a specification read by everyone is not a design method, it is a mood poll.

Beneath this particular process runs a general logic of problem-solving, older than software, and which serves at every scale. George Polya formalized it in 4 movements: understand the problem, devise a plan, carry it out, then look back to examine the solution obtained[4]. Three tools come on top, which engineering practices without always naming them. Decomposition, from the global to the particular: you cut the problem into subproblems each simpler than the whole, and regularly climb back up one floor to check that you are still solving the right problem — techniques live in layers, each layer absorbs whoever lingers in it, and the goal gets lost without this back-and-forth. Locating before solving: you do not solve a problem you have not situated, and the queen of methods is dichotomy — cut the search space in two, test which side the defect lives on, start again; this is how you isolate a breakdown, a regression in a commit history, a faulty hypothesis in a piece of reasoning. And controlled variation: change only one factor at a

time, otherwise the observed effect is attributable to nothing — the physician's differential diagnosis, the hypothesis loop of the chapter "Counting the absent". This logic is content-free: it works on a bug, a specification, an organization. That is why the chapters of this book resemble one another from one scale to the next — they are the same movements, applied to different objects.

To hold the objectives themselves, a grid served me as a starting point: the SMART method, formulated by George Doran in 1981 for writing management objectives[5]. A well-stated objective is *specific*, *measurable*, *assignable*, *realistic* and *time-related*. Specific: a concrete task, one you pick up and finish — not a global intention, and certainly nothing phrased in the vague language of politicians. The canonical counterexample, in my field: "let's turn Darktable into Lightroom". Lost from the start, not for lack of ambition, but because nothing in it is specific — neither the software, nor its history, nor its architecture, nor its users' habits are the competitor's; you have to start from where you are and name the precise improvement you are aiming for. Measurable: these are the validation criteria — the most misunderstood point, to which the next chapter is devoted. Assignable: someone carries the objective, by name; an objective nobody carries is a wish. Realistic: proportionate to the resources you actually have, including for maintenance — many good ideas die there, not from being bad, but from never having had the means of their own lifespan. Time-related: a workstream with no deadline and no time budget is a drift foretold. The grid is nothing sacred either — Doran himself wrote that not all objectives deserve the 5 criteria.

It guarantees no result; it only forces you to think about what you are doing before doing it, and that is already rare.

The contemporary objection to all this has a name: agile development. It holds in 3 grievances. The specification would belong to the "waterfall cycle", that old method where you specify the whole project first, develop next, and deliver at the end — two years later, a product conformant to the document and unsuited to the need, which changed in the meantime. You would learn more by shipping early and often than by writing documents. And the design should therefore *emerge* from the iterations, instead of being settled in advance[6]. The objection deserves better than a dismissal, for its observation is correct: full, frozen specification fails, and the short iteration is indeed the best measuring instrument we have — to ship is to measure. But you have to look at what agile practitioners do, rather than at what the slogan says: a *user story* is a use case — a defined user, a defined task, a defined benefit —, and its *definition of done* is a list of validation criteria set before development. Agility did not abolish the specification; it sliced it and moved it closer to execution — which is an improvement of the format, not a dispensation from the content. What iteration makes emerge are the solutions; the problem does not emerge: someone states it, slice by slice, or nobody states it — and iterating without a stated problem has another name: drift. Speed and direction are not the same quantity.

History ruled faster than expected — and it was not the nostalgics of the waterfall cycle who led the charge, it was the authors of the agile manifesto themselves. Dave Thomas, one of the 17

signatories, as early as 2014: the word “agile” has been “subverted to the point where it is effectively meaningless”, the community claiming it has become “an arena for consultants and vendors to hawk services and products”, the word itself “a marketing term, coopted to improve sales in the same way that words such as eco and natural are”[7]. Andy Hunt, signatory: agile has degenerated into a recitation of rules, carried by “scads of vocal agile zealots— as per the definition that a zealot is one who redoubles their effort after they’ve forgotten their aim” — and “agile methods themselves have not been agile. Now there’s an irony for you”[7]. Martin Fowler, signatory: the ordinary state of the field is *faux-agile* — “agile that’s just the name, but none of the practices and values in place” — and “the Agile Industrial Complex imposing methods on people is an absolute travesty”: the term-for-term inversion of the manifesto’s first statement, individuals and interactions over processes and tools[7]. Ron Jeffries, signatory: “it breaks my heart to see the ideas we wrote about in the Agile Manifesto used to make developers’ lives worse, instead of better” — and he calls on developers to abandon agile-by-name while keeping its values[7]. As for the “scaling” sold to large organizations, the co-creator of Scrum greeted it in one sentence: “The boys from RUP (Rational Unified Process) are back”, customization consultants included, just like RUP[7]. The academic verdict closes the file: Bertrand Meyer sorted the paradigm into the good and the ugly — on the good side, embracing change, short iterations, working code, centralized tests; on the ugly side, very precisely the abolition of the upstream: the refusal of written requirements before coding, *user stories* as a substitute for requirements, tests as a substitute for

specification, the rejection of a-priori architecture work[8]. Reread those two lists: what survives of agility is exactly what this chapter has kept of it, and what discredits it is exactly what this chapter refuses.

What to retain from this trajectory? That it is exemplary, and the chapter "The cargo cult" will give its general law: a revolt against process — 17 practitioners standing up against documentary heaviness — ended in the heaviest of liturgies: daily ceremonies, tiered certifications, dedicated roles, "scaled" frameworks — rain dances against the irreducible uncertainty of development, performed with all the more fervor as they master that uncertainty less. It is not the ideas that failed: iterate short, ship to measure, slice up the specification — all of that holds, and this chapter keeps it. It is the rite that devoured the reason, as it devours it every time a practice spreads faster than its whys. A design method does not escape the discipline it imposes on others: it too must state its use case, its criteria — and accept being judged on them.

The problem is stated, the scope chosen, the criteria written. Everything now rests on one word left hanging: *measurable*. That is the link where discipline can turn into imposture — for you can measure a great deal, and lie to yourself with numbers better than with words. Measuring without lying to yourself is the subject of the next chapter.

---

1. Attributed to Louis Srygley — "Without requirements or design, programming is the art of adding bugs to an empty

text file" — with no verifiable primary source: an aphorism of professional folklore, cited here as such. ↩

2. Fernández-Vázquez, R., Stinco, C. M., Hernanz, D. *et al.* (2013). "Colour training and colour differences thresholds in orange juice". *Food Quality and Preference*, 30(2), 320-327. https://doi.org/10.1016/j.foodqual.2013.05.018 Just-noticeable difference threshold (ΔE CIE 1976): about 2.8 before training, 1.5 after. ↩

3. Fisher, R. & Ury, W. (1981). *Getting to Yes: Negotiating Agreement Without Giving In.* Boston: Houghton Mifflin https://openlibrary.org/books/OL4260851M/Getting_to_yes. French trans. *Comment réussir une négociation.* Paris: Seuil. The central principle: negotiate over interests, never over positions. ↩

4. Polya, G. (1945). *How to Solve It.* Princeton: Princeton University Press https://doi.org/10.1515/9781400828678 (French trans. *Comment poser et résoudre un problème*, Paris: Dunod, 1965) — understand, plan, carry out, look back: the canonical grid, written for mathematics and transposable to any stated problem. ↩

5. Doran, G. T. (1981). "There's a S.M.A.R.T. Way to Write Management's Goals and Objectives". *Management Review*, 70(11), 35-36. Original letters: *Specific, Measurable, Assignable, Realistic, Time-related.* The common variants (*attainable*, *achievable*, *ambitious*…) are later mutations of the acronym. ↩

6. Beck, K. *et al.* (2001). “Manifesto for Agile Software Development” (https://agilemanifesto.org/); Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley https://books.google.com/books/about/Extreme_Programming_Explained.html?id=G8EL4H4vf7UC . The manifesto values “working software over comprehensive documentation” — while maintaining, in its principles, the continuous conversation with the business and continuous attention to design. ↩

7. Thomas, D. (2014). “Agile is Dead (Long Live Agility)”. pragdave.me, March 4, 2014 https://pragdave.me/thoughts/active/2014-03-04-time-to-kill-agile.html — “The word ‘agile’ has been subverted to the point where it is effectively meaningless”; “It became a marketing term, coopted to improve sales in the same way that words such as eco and natural are”. Hunt, A. (2015). “The Failure of Agile”. toolshed.com, May 6, 2015 https://toolshed.com/2015/05/the-failure-of-agile.html — “We have scads of vocal agile zealots—as per the definition that a zealot is one who redoubles their effort after they’ve forgotten their aim”; “agile methods themselves have not been agile. Now there’s an irony for you”. Fowler, M. (2018). “The State of Agile Software in 2018”. martinfowler.com, August 25, 2018 https://martinfowler.com/articles/agile-aus-2018.html (edited transcript of his talk at Agile Australia, Melbourne, June 2018) — “The Agile Industrial Complex imposing methods on people is an absolute travesty”.

Jeffries, R. (2018). “Developers Should Abandon Agile”. ronjeffries.com, May 10, 2018 https://ronjeffries.com/articles/018-01ff/abandon-1/ .
Schwaber, K. (2013). “unSAFe at any speed”. kenschwaber.wordpress.com, August 6, 2013 https://kenschwaber.wordpress.com/2013/08/06/unsafe-at-any-speed/ — “The boys from RUP (Rational Unified Process) are back”. All five are co-signatories of the 2001 manifesto (Schwaber is also co-creator of Scrum). ↩ ↩ ↩ ↩ ↩

8. Meyer, B. (2014). *Agile! The Good, the Hype and the Ugly*. Cham: Springer https://doi.org/10.1007/978-3-319-05155-0 — chapter 3 is titled “The Enemy: Big Upfront Anything”, and chapter 11 draws up the sorting; the “ugly” there includes the refusal of upfront written requirements, *user stories* as a substitute for abstract requirements, tests as a substitute for specification, and the rejection of a-priori architecture work. Same author as the “open/closed” of the chapter “The cargo cult”. ↩

6. Measuring without lying to yourself

The previous chapter left the validation criteria in the hands of one word: *measurable*. It is the strongest link in the method and the easiest to corrupt, because numbers enjoy a privilege: nobody suspects them. An opinion presents itself as an opinion; a number presents itself as a fact, even when it has been fabricated, cherry-picked, or hijacked. There are two main ways to lie to yourself with measurements: measuring one thing while believing you are characterizing another, and turning the measurement into a goal. And upstream of both, a primary fact that dashboard habits make people forget: a number does not exist in the wild — someone computed it, by a method and on data they chose, and often from among a set of possible methods, each with its strengths, its weaknesses, its context, more or less suited to the conditions of the problem at hand.

A number is a finished product, and it must be read as one. Before it reaches your eyes, it has passed through two floors of decisions. The first floor is the method: an equation, an aggregation procedure, a way of framing the problem, thresholds, simplifying assumptions that make the computation possible — so many choices that are anything but neutral and that decide the result as much as the measured reality does. A crash rate is computed per session or per user, and the same software comes out stable or rickety; an average task-completion time includes or excludes abandoned attempts, and the same interface comes out

fluid or discouraging; a satisfaction score adds up questions that someone chose, worded, and weighted. Metrology — the science of measurement — says nothing else: a measurement result only means something together with its method and its uncertainty, and the method is published precisely because it can be contested. That is the normal regime of physical measurement; the software dashboard, for its part, too often buries its method in a query nobody rereads. Understanding the method is therefore not expert zeal, it is the condition for knowing what you are reading — and discussing it is a right: contesting an equation, or the model it represents (or infers), is not contesting a fact. The second floor is the data: it was collected somewhere, by someone, under a protocol that has its own filters — who was polled, what was logged, what was silently lost, how the questions were asked. These biases are not always visible in the data, because some leave no trace in it: the session that crashes does not send its report, the exasperated user uninstalls before the survey. The two floors combine into a conclusion to keep in mind in front of every dashboard: when you read a number, you are already reading someone's opinion — about the right way to compute, and about the data that deserved to be collected. Which also explains where the number gets its aura: it looks powerful and objective exactly insofar as you do not know how it was obtained. And that ignorance is well guarded, because the fear of mathematics — so widely shared, so early instilled — guarantees that almost no one will dare ask to see the computation. The authority of the number does not come from the rigor of its manufacture; it comes from its manufacture being invisible and its language being intimidating. Those who measure for a living

are, of all people, the least impressed by a naked number. This does not disqualify numbers; it puts them under the regime of the chapter "Listening without obeying": a number is a testimony, open to questioning — who computed it, how, on what — and a number that cannot be audited is not a measurement, it is an argument from authority dressed up as objectivity.

That settled, what is a metric for? It serves to compare two solutions on the same criterion, and thereby to close a discussion: between two proposals, the one that gets the task done in fewer steps, in less time, with fewer errors, wins — and we move on. That is all, and it is immense: instrumented comparison is precisely the stopping rule that the chapter "The word and the thing" called for. What a metric must never become is an optimization goal. The formula is known as Goodhart's law: when a measure becomes a target, it ceases to be a good measure[1]. The mechanism holds no mystery — as soon as a number commands the reward, people optimize the number, not what it was supposed to reflect — and the canonical example comes from Donald Campbell, who studied the indicators of public policy: make the case-clearance rate the performance indicator for police officers, and you will get — among other things — officers who talk people out of filing complaints when the case looks complicated[2]. The rate goes up, the reality it measured deteriorates, and the deterioration is invisible since the indicator has become blind by construction. Worse than not measuring: measuring one thing while believing you are characterizing another, because you then make bad decisions with all the confidence of objectivity.

Let us watch the mechanism at work on the favorite metric of interface debates — the one ergonomists formalized as early as 1980 under the name *keystroke-level model*, which predicts the time of a task by adding up its elementary gestures[3]: the number of operations needed to reach the result. It has everything going for it — objectifiable, comparable, not an opinion. Yet it captures one dimension out of 3, and the full accounts often reverse the verdict. First missing dimension: safety. Darktable prides itself on a "no-click" workflow, where most operations trigger on mouse-hover over the graphical widget — fewer operations, indeed. You can also build a car with no ignition key and no seat belt: that removes steps at start-up; if it kills more passengers, is it a better design? In the old lighttable, moving the mouse could modify the metadata of an image you had not chosen — the rating you thought you were giving one photo went to the one the cursor was brushing past, and the image vanished from the current filter without your understanding why. I drew two rules from this, which cost clicks and are worth them: no data write without an explicit selection, and a selection is only explicit through a hard gesture — a click, a key; hovering is reserved for reading. Friction in front of a destructive operation is not an ergonomic flaw, it is safety equipment — what industrial design calls a *forcing function*, and Japanese industry a *poka-yoke*, a mistake-proofing device that makes the dangerous gesture impossible or costly[4]. These two rules do not protect the attentive user, who does not need them: they protect the hurried, the tired, the beginner — the one who will not be able to defend himself against my design. Second missing dimension: what I call the *cognitive price of admission* —

discovering the tool, understanding what you are manipulating, being able to predict what a gesture will do. It is not counted in clicks. The user who has been settled in for 10 years does not care whether the layout is sensible or absurd: he knows its every corner and wants to go fast; the real question is whether the organization makes sense to the newcomer, because he is the one who has to understand without knowing. Third exhibit in the record: the full accounting of the mask centralization, promised in the chapter "Listening without obeying". The complaint — "it's farther away than before" — counted clicks and cursor travel, and it counted correctly. But it counted the old design's clicks as if they were free, and they were not. The mask controls, housed at the bottom of already tall modules, appeared below the edge of the screen — usage statistics gave a first decile of screens at 960 × 540 useful pixels, and for a good half of the users, you had to click or scroll to reach them anyway. The scrolling was itself booby-trapped: in Darktable, the wheel over a module scrolls the *slider values*, not the panel content — the "free" gesture could change settings along the way. And even with that price paid, the screen still did not show the masked module and its masking parameters together. The current design reverses the distribution of costs: the visibility of the mask controls is guaranteed in full, against extra clicks and mouse travel, and the distance between the module and its mask. Add the consistency line: since the masking mechanism is identical for all modules, a common function gets centralized instead of duplicated — which is, incidentally, Lightroom's choice. Four lines of accounting — the clicks, the booby-trapped scrolling, the visibility, the consistency — where the metric saw one. And the verdict keeps the form this chapter

imposes on measurement verdicts: comparative, never final. I am not attached to this design; of all the solutions I have thought through, it is the least bad, and the file remains open to anyone who thinks they can produce a better design.

One last account before the instruments, because it shows what a feeling is worth when no measurement frames it. Several times I have optimized one atomic part of the software — a loop, a cache, a memory transfer — and said so in the release notes. In the following days, users came to report that other parts of the program, unrelated to the optimized part and untouched by any line of the change, were faster too. None of these reports rested on a stopwatch; all were sincere; and all were pleased. I call this the *feeling of speed*: give people the prior that the software should be faster, and they will feel it faster, whatever the measured timings. This is not a naivety peculiar to my users, it is a property of perception. Felt time can be manipulated without touching real time: a progress bar of the same duration, animated differently, is judged up to 11% faster[5]. The announcement is enough: participants told that a display runs at a higher refresh rate than it actually does judge their performance better, with rigorously equal reaction times — and the same phantom perception is obtained by announcing an artificial intelligence or an adaptation that do not exist[6]. The mechanism is general, and it is temporal: information received *before* the experience changes the experience itself, not just its report; received after, it changes nothing[7]. That is exactly the regime of release notes: you read them before launching the software, and learn from them what you are about to feel. Across the literature, moreover, measured

effectiveness, measured efficiency and declared satisfaction correlate only weakly[8]. The conclusion is not that the feeling does not count; it is that it does not *measure*: it measures expectation, to which performance contributed an unknown share. A speed gain is proven on the bench, on a fixed task, before and after, by the person who made the change — and first-day praise, like the first-day protest the chapter "Decide, then hold" will dissect, is a reflex, not testimony.

There remains the symmetrical temptation to faith in numbers: decreeing that what matters most — satisfaction, the pleasure of use — cannot be measured, and thereby sparing yourself the measurement. The temptation is widespread among designers of expert tools, and I know its appeal: it has apparent self-evidence on its side. But the psychometrics of usability has produced instruments, and you have to look them in the face before dismissing them. The ISO 9241-11 standard has defined usability since 1998 as the extent to which a system can be used, *by specified users*, to achieve *specified goals*, with effectiveness, efficiency and satisfaction, *in a specified context of use*[9]. The SUS scale — 10 statements rated from 1 to 5, a score out of 100 — has existed since 1986, has served in thousands of studies, and has what makes an instrument serious: norms (the empirical mean sits around 68, which is therefore the 50th percentile — a SUS score is not a percentage, and users' spontaneous "OK" corresponds to a score *below* average) and measured reliability[10]. The NASA-TLX has measured subjective mental workload since 1988 along 6 dimensions — it is very exactly the standardized instrument of my "cognitive price of admission",

validated from aviation to surgery[11]. There is even a scale for pleasure: the AttrakDiff distinguishes a product's *pragmatic* quality from its *hedonic* quality — what use provides, stimulation and identity — and shows that the two vary independently[12]. The field tests its own tools and executes its bad metrics.

What survives of the objection is narrower and more solid. First — and this is the central point — **none of these scales measures the quality of what the tool produced**. SUS measures *perceived* usability; TLX, *felt* workload; AttrakDiff, *experienced* attractiveness. None says whether the photograph is good. ISO 9241's "effectiveness" — the accuracy and completeness of the result — presupposes an external criterion of correctness, which exists in standardized form for no aesthetic production: that is the hole at the center of the edifice, and it is not about to be filled. Second, all of this is self-reported perception, and the chapter "Listening without obeying" gave the number: stated preference predicts only a quarter of measured performance — perception is not performance. Third, scores can only be read relative to a norm and a context; the ISO standard says so itself: usability is not an attribute of the product, but the outcome of an interaction in a specified context. A SUS collected from beginners says next to nothing about a trade tool in trained hands — we are back to the scope of the chapter "Stating the problem", and we understand why comparisons of expert software by panels of novices produce the rankings they produce. Fourth, a single score compresses incommensurable quantities — SUS mixes ease of learning with efficiency in use, which everything opposes in a trade tool. The honest version of the objection is therefore not

"these scales do not exist", but: **these scales measure something other than what must be judged in a work tool** — they measure the user's experience, never the user's work. Let them be used for what they measure: comparing two versions of the same tool, on the population targeted by the specification, with the score as an instrument of comparison — never as a target, Goodhart stands watch — and the judgment on the work remains to be built by other means: the validation criteria of the chapter "Stating the problem", and physics when it has a say.

Measuring without lying to yourself thus comes down to a few rules. Ask for the method and the provenance of the data before believing the number — an unauditable number proves nothing. Know what the metric measures — and ask again regularly, because the answer changes when usage changes. Compare, never optimize. Complete every account before returning the verdict: safety and the price of admission show up in no click count. Grant the instruments what they measure, and refuse them what they do not measure. There remains a question this chapter has left open: every number was collected on *someone* — a sample, never the population —, and the difference between a significant gap and noise, and the difference between those who are counted and those who are missing, are questions of statistics. They deserve their own chapter: counting the absent.

---

1. Goodhart, C. A. E. (1975). "Problems of Monetary Management: The U.K. Experience", in *Papers in Monetary Economics*, vol. I. Sydney: Reserve Bank of Australia https://doi.org/10.1007/978-1-349-17295-5_4 . The famous

formulation — "When a measure becomes a target, it ceases to be a good measure" — is from Strathern, M. (1997). "'Improving ratings': audit in the British University system". *European Review*, 5(3), 305-321 https://doi.org/10.1002/(SICI)1234-981X(199707)5:3%3C305::AID-EURO184%3E3.0.CO;2-4 . ↩

2. Campbell, D. T. (1979). "Assessing the impact of planned social change". *Evaluation and Program Planning*, 2(1), 67-90 https://doi.org/10.1016/0149-7189(79)90048-x : the more a quantitative social indicator is used for decision-making, the more subject it is to corruption pressures and the more it corrupts the process it was meant to monitor — with, explicitly, the example of police statistics. ↩

3. Card, S. K., Moran, T. P. & Newell, A. (1980). "The Keystroke-Level Model for User Performance Time with Interactive Systems". *Communications of the ACM*, 23(7), 396-410 https://doi.org/10.1145/358886.358895 ; developed in *The Psychology of Human-Computer Interaction*, Hillsdale: Erlbaum, 1983 https://doi.org/10.1201/9780203736166 — the exact formalization of the gesture count this chapter criticizes. ↩

4. Forcing functions: Norman, D. A. (2013). *The Design of Everyday Things*, revised ed., chap. 4 (interlocks, lock-ins, lockouts) https://www.hachettebookgroup.com/titles/don-norman/the-design-of-everyday-things/9780465050659/ . The *poka-yoke*: Shingo, S. (1986). *Zero Quality Control: Source Inspection and the Poka-Yoke System*. Cambridge, MA:

Productivity Press https://doi.org/10.4324/9780203733639 — mistake-proofing at the source, in industry. ↩

5. Harrison, C., Yeo, Z. & Hudson, S. E. (2010). "Faster Progress Bars: Manipulating Perceived Duration with Visual Augmentations". *Proceedings of CHI 2010*, 1545-1548 https://doi.org/10.1145/1753326.1753556; Harrison, C., Amento, B., Kuznetsov, S. & Bell, R. (2007). "Rethinking the Progress Bar". *Proceedings of UIST 2007*, 115-118 https://doi.org/10.1145/1294211.1294231 — at identical duration, the progress function and the animation change perceived duration. In the same vein, on value rather than speed: Buell, R. W. & Norton, M. I. (2011). "The Labor Illusion". *Management Science*, 57(9), 1564-1579 https://doi.org/10.1287/mnsc.1110.1376. ↩

6. Bosch, E., Welsch, R., Ayach, T., Katins, C. & Kosch, T. (2024). "The Illusion of Performance: The Effect of Phantom Display Refresh Rates on User Expectations and Reaction Times". *CHI EA '24*, 1-6 https://doi.org/10.1145/3613905.3650875; Kosch, T., Welsch, R., Chuang, L. & Schmidt, A. (2022). "The Placebo Effect of Artificial Intelligence in Human-Computer Interaction". *ACM Transactions on Computer-Human Interaction*, 29(6), art. 56 https://doi.org/10.1145/3529225; Denisova, A. & Cairns, P. (2015). "The Placebo Effect in Digital Games: Phantom Perception of Adaptive Artificial Intelligence". *Proceedings of CHI PLAY 2015*, 23-33 https://doi.org/10.1145/2793107.2793109. ↩

7. Lee, L., Frederick, S. & Ariely, D. (2006). "Try It, You'll Like It". *Psychological Science*, 17(12), 1054-1058

https://doi.org/10.1111/j.1467-9280.2006.01829.x — information given before tasting changes the experience; given after, it does not. The neural correlate: Plassmann, H., O'Doherty, J., Shiv, B. & Rangel, A. (2008). "Marketing Actions Can Modulate Neural Representations of Experienced Pleasantness". *PNAS*, 105(3), 1050-1054 https://doi.org/10.1073/pnas.0706929105; and on actual, not merely perceived, performance: Shiv, B., Carmon, Z. & Ariely, D. (2005). "Placebo Effects of Marketing Actions". *Journal of Marketing Research*, 42(4), 383-393 https://doi.org/10.1509/jmkr.2005.42.4.383. ↩

8. Hornbæk, K. & Law, E. L.-C. (2007). "Meta-analysis of Correlations Among Usability Measures". *Proceedings of CHI 2007*, 617-626 https://doi.org/10.1145/1240624.1240722 — across 73 studies, effectiveness, efficiency and satisfaction correlate only at r ≈ 0.2-0.3. ↩

9. ISO 9241-11:2018, *Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts* https://www.iso.org/standard/63500.html, § 3.1.1; effectiveness (3.1.12), efficiency (3.1.13) and satisfaction (3.1.14) are defined separately there. First version: 1998. ↩

10. Brooke, J. (1996). "SUS: A 'quick and dirty' usability scale", in Jordan, P. W. *et al.* (eds.), *Usability Evaluation in Industry*. London: Taylor & Francis https://doi.org/10.1201/9781498710411-35. Norms and reliability (α ≈ 0.91 over 2,324 questionnaires): Bangor, A., Kortum, P. T. & Miller, J. T. (2008). "An Empirical Evaluation of the System Usability Scale". *International Journal of*

Human-Computer Interaction, 24(6), 574-594 https://doi.org/10.1080/10447310802205776; percentiles and adjective equivalents: Sauro, J. & Lewis, J. R. (2012). *Quantifying the User Experience*. Waltham, MA: Morgan Kaufmann https://www.sciencedirect.com/book/9780123849687/quantifying-the-user-experience. ↩

11. Hart, S. G. & Staveland, L. E. (1988). "Development of NASA-TLX: Results of Empirical and Theoretical Research", in Hancock, P. A. & Meshkati, N. (eds.), *Human Mental Workload*. Amsterdam: North-Holland, 139-183 https://doi.org/10.1016/s0166-4115(08)62386-9. The twenty-year retrospective: Hart, S. G. (2006). "NASA-Task Load Index (NASA-TLX); 20 Years Later". *Proceedings of the HFES Annual Meeting*, 50(9), 904-908 https://doi.org/10.1177/154193120605000909. ↩

12. Hassenzahl, M., Burmester, M. & Koller, F. (2003). "AttrakDiff: Ein Fragebogen zur Messung wahrgenommener hedonischer und pragmatischer Qualität", in *Mensch & Computer 2003*. Stuttgart: Teubner, 187-196 https://doi.org/10.1007/978-3-322-80058-9_19. ↩

7. Counting the absent

The previous chapter stopped at the edge of a question: every number was collected on someone. That someone is a sample — never the whole population — and the sample raises two distinct questions, which this chapter takes in order: how to summarize those who are in the numbers; and how to think about those who are not. The second is the harder one, because the absent leave no trace in the data — that is their definition — and yet they are, often, the most important line in the table.

Let us begin with the present. The first summarizing reflex is the mean, and it is almost always the wrong one: the mean only says something about a roughly Gaussian distribution, and a handful of extreme values is enough to skew it. On the amount of RAM my users have, some twenty machines with 64 GB shift the mean of the whole sample — the resulting number no longer describes anyone. What you are actually after through the mean is something else: where the bulk of the users sit. The answer fits in 3 numbers — the median, the first decile (P10) and the last one (P90): between P10 and P90 you hold 80% of the users, and they are the ones decisions are made on. Because that is what a technical decision is, read statistically: a choice of whom it includes and whom it excludes. "The reference screen will be 1920 × 1080" does not state a preference: it states how many machines fall below the bar, and P10 gives the number. In another register, the distribution also delivers the service promised in the

chapter "The root and the symptom": it shows what normal variation looks like. Before hunting for the cause of a deviation, you check whether it leaves the ordinary range — everything that happens within the mean ± n standard deviations is the system's own noise; what leaves it deserves a *why*[1].

Next, the sample is not the population, and it is never a neutral extract of it: it has been filtered, and the filters are rarely the ones you think. Free software filters first on computer skills, and it takes numbers to measure how brutal that filter is. The OECD's PIAAC survey — 215,942 adults tested across 33 countries — gives the scale: about a quarter of adults cannot use a computer at all, two thirds never get past one-step tasks, and only 5% reach the level of complex tasks[2]; the International Telecommunication Union counts, across 91 countries, roughly half the population unable to download and install a piece of software[3]. Asking someone to open a command line therefore means having already turned away more than 90% of the world — before the first survey question. My own surveys bear the mark of this filter: among Darktable users, more than 35% hold a master's degree or a doctorate, against less than 14% in the general population[4]. The methodological consequence belongs to the chapter "Measuring without lying to yourself": a blind satisfaction survey, one that does not ask *who* is answering, does not measure satisfaction — it measures who uses the software. The results say as much about the sociology of the sample as about the qualities of the tool; you need two-dimensional studies, satisfaction *and* the who, otherwise you mistake the profile of your survivors for a verdict on your work.

There, the word is out: survivors. The most vicious trap in usage statistics is that, by construction, your data contains only those who use the software — never those who gave up, never those who did not come. Worse, among the users, those who will cooperate with surveys are the fraction most motivated and most emotionally invested in the software project. Reasoning about them must be inverted, and the inversion has a famous precedent. In 1943, the statistician Abraham Wald, working for American military aviation, developed a method for estimating aircraft vulnerability from the damage of only those planes that had *returned* from missions: since the downed ones are missing from the data, the zones where the survivors are never hit are precisely the ones where hits bring planes down — that is where the armor should go, not where the survivors carry their impacts[5]. The anecdote as it is usually told — Wald turning around, with one sentence, the officers who wanted to armor the riddled zones — is a late reconstruction; the method, though, is documented in his memoranda, and the method is what counts. Its software translation is direct: by dint of improving the tool for those who stayed, you optimize it for the profile that survived the filter — and make it more off-putting still for everyone else. User feedback lives in the same bubble: those who hated it left, those who remain love the tool as it is, and their unanimity does not prove the tool is good — it proves the filter has finished its work and a kind of homeostasis has been reached.

The absent have another factory, a more active one: the self-fulfilling prophecy, described by Robert Merton — a false definition of the situation that evokes the behaviors which will

make it come true after a while[6]. In software, it has a well-oiled cycle: a badly designed feature is little used; its usage numbers, being low, justify never improving it; it stays badly designed, hence little used — and the numbers are right for the wrong reasons. My statistics provide a textbook case: 37% of my users own a graphics tablet, 6% use one in the software. Then again, 10% own a MIDI controller, and 2% use one in the software. Given that MIDI controllers are mostly dedicated to music, and that Darktable's support for them is as complete as one can imagine, the 2% usage would rather indicate a lack of interest. Given that the graphics tablet is a device specifically dedicated to image processing and that Darktable uses drawn masks, the 6% usage would rather indicate defective usability. In both cases, the usage rate brought back to the ownership rate (1 in 5 and 1 in 6) already gives a more complete indicator than raw usage, but it remains insufficient to conclude, and the "owns without using" case must therefore be investigated specifically. Hence the general rule, which is the elementary scientific posture: a number is not to be contemplated, it is to be confronted. You start from a hypothesis — hence from a prejudice, owned as such —, you derive a prediction from it — "if this is true, we should observe this" —, and you check whether the data confirms or refutes it[7]. Collecting statistics presupposes already having an idea of what you want to know; a dashboard you stare at without a question teaches nothing, it reassures.

The instrument itself has a bad reputation, and it must first be put in perspective. Opt-in telemetry leans toward the configurations of the most engaged users — the only ones who leave collection

enabled. This is exactly the survivorship bias: it describes faithfully who stayed, and nothing of who left. From this I draw the hierarchy that holds for every usage dashboard: telemetry is reliable for *ranking within* the served population — which format, which resolution, which graphics card weighs most in it — never for setting *its boundary*; deciding whom the tool should serve is a matter of comparison against a reference population, not of counting those already there. That settled, I equipped my software with usage statistics, and the first rule was transparency: say what is collected, why, and publish what is learned from it — users may accept to participate more readily when they see what the accounts are for. The yield is real. Knowing that more than 43% of the machines have no graphics acceleration is enough to cool the enthusiasm for architectures that bet everything on the graphics card: for those 43%, such software would not be an option. Telemetry ranks the pile of problems instead of enduring it, and it also protects the maintainer — when a noisy demand represents 5% of the sample, the number puts the discussion back at its size. This defensive use has its limit, and it is exactly Merton's prophecy: "you are 5%" can be the measure of a marginal need or of a flaw that drove the others away — the ownership rate, the comparison against a reference population, the gap between owning and using, are the controls that settle it. The instrument never exempts you from the hypothesis.

There remains an objection of principle, which has the zeitgeist on its side, and it must be stated in full so that it gets its chance. Here it is: the designer's judgment is fallible, biased, unverifiable,

so let us collect usage data and do what it says: keep what is used, remove what is not, prioritize what is asked for, and settle debates at the dashboard rather than by authority. This is the doctrine called *data-driven*, so dear to technocrats: the data decides, the human executes. The whole chapter has already answered it piece by piece: the data contains only the survivors of a filter it does not document; it is read through hypotheses it does not supply; and the decision of who counts is a choice of values that no data will make — that is the scope of the chapter "Stating the problem". Sometimes, the intelligent decision goes against the numbers. A category that weighs 2% in the sample can weigh 15 or 20% in reality, if it was filtered out, notably by complexity — the difficult installation, the repository to configure, the antivirus declaring the executable infected. Those absent are missing for a bad reason, and reaching out to them is a decision that will never have a majority in the data, since they are not in it. If people are absent because the tool does not interest them, fine; if they are absent because they understood none of it, that is a design problem — and it will show up in no dashboard. Noticing a need nobody testifies to is the most costly gesture of the trade: nothing rewards it, no number calls for it, and yet it is where everything begins. Being guided by the data beats being guided by your navel, but it is not a waiver of judgment. Data informs the decision; it does not make it. And any decision-making imposes a hierarchy of values which imposes a hierarchy of priorities: all of them must be clearly stated and owned.

Summarize the present by their distribution, know your sample's filter, reason about the absent by inversion, break the prophecies before they fulfill themselves, confront every number with a hypothesis: that is the toolkit. There remains the gesture that all this apparatus prepares and that no apparatus performs — deciding, then holding to what you have decided. That is the subject of the next chapter.

---

1. A rule of thumb, to be handled as such: the rigorous analysis of a process's stability uses Shewhart's and Deming's control charts — limits at ±3 standard deviations around the mean of a process *tracked over time*, not the deciles of an instantaneous distribution (Shewhart, W. A. (1931). *Economic Control of Quality of Manufactured Product*. New York: Van Nostrand https://archive.org/details/in.ernet.dli.2015.150272 ; Deming, W. E. (1986). *Out of the Crisis*. Cambridge, MA: MIT CAES https://doi.org/10.7551/mitpress/11457.001.0001). The P10-P90 range gives a useful approximation for usage data without a history, but it describes dispersion, not stability: a system can drift wholesale while keeping the same range. ↩

2. OECD (2016). *Skills Matter: Further Results from the Survey of Adult Skills*. Paris: OECD Publishing https://doi.org/10.1787/9789264258051-en — "problem solving in technology-rich environments" module (PIAAC; 215,942 adults, 33 countries): 26% of adults cannot use a computer, 69% do not get past level 1 (one-step tasks), 5% reach level 3. Summary: Nielsen, J. (2016). "The Distribution

of Users' Computer Skills". Nielsen Norman Group https://www.nngroup.com/articles/computer-skill-levels/ . ↩

3. International Telecommunication Union (2021). *Measuring digital development: Facts and Figures 2021*. Geneva: ITU https://www.itu.int/itu-d/reports/statistics/facts-figures-2021/ — across 91 countries, 48.8% of the population does not know how to download and install software. ↩

4. Surveys conducted by the author among Darktable users in 2020 (n = 1,101) https://eng.aurelienpierre.com/2023/01/who-are-the-Darktable-users-in-2020/ and 2022 (n = 1,253) https://eng.aurelienpierre.com/2026/07/who-are-the-Darktable-users-in-2022/; anonymized data and processing code published with the results (eng.aurelienpierre.com). General population: OECD, educational attainment of 25-64 year-olds. ↩

5. Wald, A. (1943). *A Method of Estimating Plane Vulnerability Based on Damage of Survivors*. Statistical Research Group, memoranda 85-126; repr. CNA, 1980 https://www.cna.org/analyses/1980/a-method-of-estimating-plane-vulnerability . Modern analysis: Mangel, M. & Samaniego, F. J. (1984). "Abraham Wald's Work on Aircraft Survivability". *JASA*, 79(386), 259-267 https://doi.org/10.1080/01621459.1984.10478038 . On the legend in the popular tale: Casselman, B. (2016). "The Legend of Abraham Wald". AMS Feature Column https://www.ams.org/publicoutreach/feature-column/fc-2016-06 — the scene and the famous red-dotted diagram appear nowhere in the memoranda. ↩

6. Merton, R. K. (1948). “The Self-Fulfilling Prophecy”. *Antioch Review*, 8(2), 193-210 https://doi.org/10.2307/4609267 . ↩

7. Popper, K. (1934). *Logik der Forschung*. Vienna: Springer; Engl. transl. *The Logic of Scientific Discovery*. London: Hutchinson, 1959 https://doi.org/10.4324/9780203994627 — knowledge advances by conjectures exposed to refutation, never by contemplation of the data. ↩

8. Decide, then hold

The whole apparatus of the preceding chapters — the problem stated, the criteria written down, the measurements taken — prepares a move that no apparatus performs: deciding. Deliberation has a shape that anyone who has ever designed in a group will recognize: you pick a solution, discuss it, and it starts looking worse; you pick another, discuss it, and it fades in turn; and step by step you circle back to the first idea. This loop is not a failure; it is the normal topology of a solution space where nothing is perfect — but you have to know how to recognize it: once you have gone around the question 3 times, the discussion teaches nothing more, and it is time to decide. This is where the objectivation of the chapter "Measuring without lying to yourself" renders its service — measurable criteria let you close what tastes would leave open — under one condition of hygiene: never twist a metric away from what it means in order to prove yourself right. And the bar for deciding is lower than you think: design being iterative, a decision does not have to be optimal right away — it has to be better than the present state, and reversible if it is not.

Deciding, though, is only half the move. The other half begins the next morning, because where tools are concerned, a decision is tested in real use — nothing replaces use — and the full-scale test has a social cost every maintainer knows: every change triggers protests, whatever its quality. The phenomenon has a name,

change aversion, and a literature that takes its mechanisms apart. Status quo bias: with options otherwise equal, we prefer what exists, simply because it exists[1]. Loss aversion: an interface change *takes away* something we had — and losses weigh psychologically more than equivalent gains[2]. The mere-exposure effect: the familiar enjoys a likability advantage over the new, until the new has accumulated its own familiarity[3]. And one very real cost: switching tools invalidates acquired automatisms, and the relearning cost is roughly proportional to the know-how lost[4]. Four mechanisms, all independent of the quality of the new design. The consequence is brutal for anyone trying to judge a change: first-day protest tells you close to nothing. The textbook cases confirm it. When Facebook launched its News Feed in 2006, the protest reached a sizable fraction of the user base — while usage, meanwhile, kept climbing: revealed preference contradicted declared aversion. When Microsoft replaced Office's menus with the Ribbon, the reception was broadly favorable — *except* among the most experienced users of the old interface[5]: resistance concentrates where the capital of habit is largest, not where the design is worst.

I lived this theorem in miniature. The day I removed the module search box — replaced by a contextual search behind a keyboard shortcut: Ctrl+P, then the module's name — the protests were universal. Yet you had to type the module's name anyway: the keyboard was already involved, so wiring the search to a keyboard shortcut imposed no extra trip from mouse to keyboard. Users had lost nothing — same function, presented differently — and the interface freed up a full line of vertical

space; the function remained fully discoverable, since the shortcut is announced and the search accessible: what had been removed was not the ability to search, but a permanent *box* cluttering the interface. It took explaining it two or three times, and everyone came around to judging the solution more efficient; but I had to hold, for a few days, against protests coming — as with the Ribbon — from the most seasoned users, that is, from those who had the most automatisms to relearn and the least to gain. Had I caved on the first evening, the right solution would never have seen the light of day.

How, then, to distinguish the protest of habit — which passes — from the genuine design flaw — which stays? There is a criterion, and it is remarkably elegant: **change aversion cannot exist in someone who never knew the old version**[6]. A new user has no status quo to defend, no loss to feel, no automatism to relearn: if he stumbles, it is not habit — it is the design. The diagnosis follows: the difficulty that users *without history* reproduce, or the one that persists among veterans after a genuine re-acclimation, is a real flaw; the protest that decays down to a satisfaction level better than before was only the friction of relearning. And re-acclimation has its orders of magnitude: a few days of honest use before an opinion is admissible — the time it takes for the mere-exposure penalty to dissipate —, a few weeks before the verdict is final; the full re-automation of gestures is slower still — habit formation plateaus at a median of about sixty days[7]. Hence the protocol that makes the test socially possible, and which is a contract: users grant the change a real, sufficiently long trial — not an indignant first-evening skim —; the designer promises the

rollback if the experiment is not conclusive, and keeps that promise. The rollback is not a humiliation, it is a clause of the contract — and it is what entitles you to demand the trial. Nor is it one more courtesy: promising the rollback is acknowledging that you are experimenting on someone else's work. Keeping your choices reversible until you have proof that they do the user good is not a cautious temperament; it is a duty.

Two traps await at the exits of this protocol. The first is the sunk-cost fallacy: keeping a bad solution *because* you spent 3 months on it. The time spent is lost whatever you decide; the only open question is the cost from here on, and a bad solution kept in place loses you more still[8]. Digging in is not constancy; it is paying twice. The second trap is symmetric: daring nothing, on the grounds that it ought to work on the first try. Nothing gets created without the freedom to be wrong. Production software is not an open-air laboratory, but it demands a latitude for experimentation — bounded by the protocol above, precisely so that it remains something users can be asked to accept. And the opposite of refusing change is not the headlong rush: changing for the pleasure of the new, the disruptive, the innovative is inflicting the 4 mechanisms of aversion on your users with nothing in return. A change is justified by an account — chapter "Measuring without lying to yourself" —, never by its date.

That leaves the last word of the title: hold. A closed decision must stay closed, because a decision that reopens at will is not a decision, and because perpetually reopened discussions are the bottomless pit that the chapter "The word and the thing" was trying to avoid. The rule for reopening is the same as the rule for

closing: something new — a new data point, a new use case, a flaw the protocol has confirmed — reopens the file; one more opinion does not. Faced with the same discussion coming back for the nth time with nothing new, once the explanation has been given, the protocol offered, the trial refused, it is legitimate to close — on the strength of experience and command of the record, and on the strength of the work you provide and your interlocutor does not. It is not the elegant way; it is the circuit breaker, and a circuit breaker serves only as a last resort — used routinely, it would turn authority into a mode of government, which the last part of this book fights precisely. Between authoritarianism and timidity, the ridge line is narrow; you hold it by publishing your criteria, promising your rollbacks, and reserving the use of force for dead ends.

The final objection is known in advance: "the user is always right." The book has already apportioned the rights — the user has authority over his pains, never over the solutions (chapter "Listening without obeying") —, and this chapter adds the temporal piece: first-day protest is not testimony of use, it is a loss reflex; testimony begins after the honest trial, and Facebook's News Feed, like Office's Ribbon, shows what declared aversion is worth against revealed preference. The user is always right about what hurts him; he is rarely right, on the first evening, about what has just changed.

State, measure, count, decide, hold: 5 moves, and none of them required inventing a bespoke tool — each leaned on an instrument already broken in elsewhere, borrowed from those who had broken it in before us. A few remain in the box that no

chapter had its own reason to summon, and that you must nonetheless know before closing this part of the book — for a discipline that ignores its own toolkit improvises twice as much as it should. That is the object of the next chapter.

---

1. Samuelson, W. & Zeckhauser, R. (1988). "Status Quo Bias in Decision Making". *Journal of Risk and Uncertainty*, 1(1), 7-59. https://doi.org/10.1007/bf00055564 ↩

2. Kahneman, D. & Tversky, A. (1979). "Prospect Theory: An Analysis of Decision under Risk". *Econometrica*, 47(2), 263-291 https://doi.org/10.2307/1914185 — the loss/gain asymmetry (loss aversion), applied to interface change: the new "takes away what you had." ↩

3. Zajonc, R. B. (1968). "Attitudinal Effects of Mere Exposure". *Journal of Personality and Social Psychology*, 9(2, Pt. 2), 1-27. https://doi.org/10.1037/h0025848 ↩

4. Singley, M. K. & Anderson, J. R. (1989). *The Transfer of Cognitive Skill*. Cambridge, MA: Harvard University Press https://openlibrary.org/works/OL4639862W — the transfer cost grows with the share of invalidated automatisms: real, temporary, proportional to the lost overlap. ↩

5. Facebook News Feed (September 2006): an episode documented by the press of the day — massive, highly visible protest, rising usage. Office Ribbon: Dostál, M. (2010). "User Acceptance of the Microsoft Ribbon User Interface". *Proceedings DNCOCO 2010*, 143-149. https://wseas.us/e-

library/conferences/2010/Faro/DNCOCO/DNCOCO-25.pdf ↩

6. Sedley, A. & Müller, H. (2013). "Minimizing Change Aversion for the Google Drive Launch". *CHI '13 Extended Abstracts*, 2351-2354 https://doi.org/10.1145/2468356.2468767 — defines change aversion, tracks its trajectory to a "new baseline state", and states the differential diagnosis: "If the product is very difficult to learn or use, it cannot be because of change aversion, but due to fundamental design flaws, or to expectations exceeding reality." ↩

7. Lally, P., van Jaarsveld, C. H. M., Potts, H. W. W. & Wardle, J. (2010). "How are habits formed: Modelling habit formation in the real world". *European Journal of Social Psychology*, 40(6), 998-1009 https://doi.org/10.1002/ejsp.674 — automaticity plateau reached at a median of 66 days (range: 18 to 254). The trial durations given here are field practice, not a threshold drawn from the literature. ↩

8. Arkes, H. R. & Blumer, C. (1985). "The Psychology of Sunk Cost". *Organizational Behavior and Human Decision Processes*, 35(1), 124-140. DOI ↩

9. Tooling up without submitting

The preceding chapters laid out a discipline: state the problem, measure without lying to yourself, count the absent, decide then hold. None of these moves was invented here, and that is good news: design is a method that has been taught for half a century, with its textbooks, its tools and its measured results. The bad news is that almost nobody in free software has opened them. There, people design the way they tinker on a Saturday afternoon: by intuition, by enthusiasm, by whoever spoke last. This chapter therefore takes inventory of the toolbox — not everything that exists, but what a designer of work tools should know before claiming to do better. And it does so under a rule the chapter "Measuring without lying to yourself" imposes: each tool will be weighed at its level of evidence. Some rest on clinical trials; others on a laboratory experiment; others still are nothing but convenient vocabulary draped in authority. Confusing them is already submitting.

Start with the bedrock, because you need to know it exists: systematic design is an established academic corpus. The German school codified it as early as the 1970s — Pahl and Beitz, the "bible" of mechanical design across the Rhine, splits every project into 4 phases: clarify the task, design the principle, embody the solution, define the detail[1]. The American school turned it into the standard textbook of engineering and business schools — Ulrich and Eppinger, 8 editions since 1995: identify the needs,

establish the specifications, generate several concepts, select them against criteria, and only then develop[2]. Reread this part of the book with those tables of contents in hand: stating the problem is task clarification; measuring and counting are the specifications and their criteria; deciding then holding is concept selection. This book does not invent its discipline — it recovers it, and the fact deserves to be said bluntly: when a software project improvises its design in a discussion thread, on a forum, it is not ahead of these "industrial" methods; it is half a century behind them.

From this corpus, the first tool to keep is the concept selection matrix, known as the Pugh matrix[3]. The principle: you pick a reference concept — the *datum*, often the existing solution —, line up the candidate concepts in columns and the specification's criteria in rows, and each candidate is judged, criterion by criterion, better (+), same (S) or worse (–) than the reference. It is almost always deformed: the original method weights nothing and computes nothing — the weighted versions are late grafts. Its strength is not to produce a score; it is to organize a convergence: on the next round, you hybridize the candidates, take the emerging winner as the new reference, and start again. The matrix does not decide in the team's place; it forces the discussion to pass through each criterion, one by one, against a shared reference — exactly what a brainstorm of opinions never does. It is a tool for structured thinking, not an objectivity machine, and whoever sells you the contrary is selling the deformation, not the method.

The second tool has prevented deaths, and can prove it: the checklist. An intensive-care physician reduced a procedure everyone knew — placing central lines — to a list of 5 elementary steps, all common knowledge, all regularly skipped; tested in his hospital, then deployed across the ICUs of an entire American state, the list made the infections everyone believed to be the unavoidable price of care collapse[4]. The World Health Organization generalized it: 19 checkpoints around a surgical operation, tried in 8 cities on 4 continents — mortality down from 1.5% to 0.8%, major complications from 11% to 7%[4]. The mechanism deserves to be understood, because there is nothing bureaucratic about it: a checklist is institutional memory taken out of people's heads — the list of known failure modes, kept up to date, consulted at the moment human attention is least reliable. And the guardrail is already written in the chapter "Measuring without lying to yourself": the list alerts, it does not absolve. A checklist whose boxes you tick to be in compliance, instead of looking for what it detected, is dead — the chapter "The transparent tool" will give a quantified example.

The third tool attacks the polite silence that surrounds decisions, and the chapter "Decide, then hold" showed what that silence costs. The *premortem* inverts the review exercise: instead of asking "what could go wrong?" — a question that invites everyone to spare the project and its author —, you declare: "the project has failed, completely; tell us why." The trick is not rhetorical, it is documented: placing yourself *after* the imagined event markedly improves the ability to identify its causes — the decision psychologists' prospective hindsight[5]. Above all, the

premortem grants a permission: in a room where the architect has already decided, saying "here is what looks fragile to me" is an attack; telling why the dead project died is a team game. It is the cheapest tool in this chapter — one hour, zero software.

The fourth tool is a word, and it is worth a method: *satisficing*. Herbert Simon — the same Simon as the sciences of the artificial in the chapter "The word and the thing" — showed that real organisms, with their bounded capacities, do not seek the optimum: they seek the sufficient, and stop there[6]. What looks like laziness is the only practicable rationality, and it answers the structural defect of design problems: the chapter "The word and the thing" said, with Rittel, that they have no stopping rule. A design can always be polished further; that is why the sufficiency threshold must be written *into the specification*, before starting — past this measured contrast, past this response time, we stop. Optimizing beyond the threshold is not rigor; it is procrastination dressed up as mathematics; and writing no threshold at all is condemning yourself to settle every stop by power struggle or by exhaustion.

That leaves the tools you must mention because they are everywhere, and must weigh because their aura exceeds their evidence. The MoSCoW grid — *Must, Should, Could, Won't* — sorts requirements into 4 urgencies; it was born with an Oracle practitioner in 1994, adopted by an agile-method consortium, taught in every training course — and never validated: no controlled study at its birth, and the only recent empirical study to tackle it concludes cautiously that it "may not be as effective and precise as previously believed"[7]. Verdict: a convenient

vocabulary for understanding one another in a meeting — that is already something —, not a proven method, and above all not an argument. TRIZ, the "theory of inventive problem solving," is more impressive: a Soviet engineer combed through tens of thousands of patents to extract 40 recurring principles of invention[8]. The corpus is real and the idea — look for solution patterns already proven elsewhere — is sound; but the systematic reviews find that practice uses only two or three basic tools out of the whole edifice, and that no controlled validation ever followed the stated ambitions; the success rates that circulate in training brochures rest on nothing[8]. Verdict: a mine of prompts for thinking by analogy, to be looted freely — and a church not to join, certification included.

This final ranking gives the chapter its law. A design tool is worth the mechanism it structures — a discussion (Pugh), a memory (checklist), a dissent (premortem), a stop (satisficing) — and the evidence that backs it, in that order. It is never worth its acronym, its certification industry or the number of its adherents: when the paid training prospers faster than the evidence, what you are buying is no longer a method, it is a membership. This part of the book therefore closes on this warning, which announces another: the chapter "The cargo cult" will show what becomes of a rule — even an excellent one — when it is recited without its reason. Equipped with the toolkit, we can descend where almost nobody looks: beneath the interface, into the machine. That is the object of the part "The machine".

1. Pahl, G. & Beitz, W. (2007). *Engineering Design: A Systematic Approach*, 3rd ed. (with J. Feldhusen and K.-H. Grote; trans. K. Wallace & L. Blessing). London: Springer https://doi.org/10.1007/978-1-84628-319-2 (German original *Konstruktionslehre*, 1977). The book codifies the tradition known as VDI 2221: task clarification, conceptual design, embodiment design, detail design. ↩

2. Ulrich, K. T. & Eppinger, S. D. (2020). *Product Design and Development*, 7th ed. New York: McGraw-Hill (1st ed. 1995; 8th ed. 2026 with M. C. Yang) https://search.worldcat.org/title/Product-design-and-development/oclc/904505863 . ↩

3. Pugh, S. (1981). "Concept Selection: A Method That Works", in Hubka, V. (ed.), *Proceedings of the International Conference on Engineering Design (ICED 81)*, Rome. Zurich: Heurista, pp. 497-506; developed in *Total Design: Integrated Methods for Successful Product Engineering*. Wokingham: Addison-Wesley, 1991 https://search.worldcat.org/title/Total-design-:-integrated-methods-for-successful-product-engineering/oclc/878735886 . The original method is qualitative (+/S/− against a *datum*, "controlled convergence" through successive rounds); the numerical weightings are later extensions, foreign to the founding text. ↩

4. Gawande, A. (2009). *The Checklist Manifesto: How to Get Things Right*. New York: Metropolitan Books https://search.worldcat.org/title/The-checklist-manifesto-:-how-to-get-things-right/oclc/546868192 . The two underlying

studies: Pronovost, P. *et al.* (2006). "An Intervention to Decrease Catheter-Related Bloodstream Infections in the ICU". *New England Journal of Medicine*, 355(26), 2725-2732 https://doi.org/10.1056/NEJMoa061115 — the statewide Michigan deployment (the initial pilot, run at Johns Hopkins, is Berenholtz, S. M. *et al.*, 2004, *Critical Care Medicine*, 32(10) https://doi.org/10.1097/01.ccm.0000142399.70913.2f); and Haynes, A. B. *et al.* (2009). "A Surgical Safety Checklist to Reduce Morbidity and Mortality in a Global Population". *New England Journal of Medicine*, 360(5), 491-499 https://doi.org/10.1056/NEJMsa0810119 — eight hospitals, eight cities, four continents. ↩ ↩

5. Klein, G. (2007). "Performing a Project Premortem". *Harvard Business Review*, 85(9), 18-19 https://hbr.org/2007/09/performing-a-project-premortem. The experimental grounding: Mitchell, D. J., Russo, J. E. & Pennington, N. (1989). "Back to the Future: Temporal Perspective in the Explanation of Events". *Journal of Behavioral Decision Making*, 2, 25-38 https://doi.org/10.1002/bdm.3960020103 — the roughly 30% gain in identifying causes is the figure Klein attributes to this study. ↩

6. Simon, H. A. (1956). "Rational Choice and the Structure of the Environment". *Psychological Review*, 63(2), 129-138 https://doi.org/10.1037/h0042769: "organisms adapt well enough to 'satisfice'; they do not, in general, 'optimize'" (p. 129). The companion article: "A Behavioral Model of

Rational Choice". *Quarterly Journal of Economics*, 69(1), 1955, 99-118 https://doi.org/10.2307/1884852 . ↩

7. Clegg, D. & Barker, R. (1994). *CASE Method Fast-Track: A RAD Approach*. Wokingham: Addison-Wesley https://search.worldcat.org/title/CASE-method-fast-track-:-a-RAD-approach/oclc/59952336 . A heuristic devised at Oracle, adopted by the DSDM consortium in the early 2000s, without experimental validation; the recent critical study: "Assessing the Effectiveness of MoSCoW Prioritization in Software Development", *EAI Endorsed Transactions on Internet of Things*, 2024 https://doi.org/10.4108/eetiot.6515 . ↩

8. Altshuller, G. (1999). *The Innovation Algorithm: TRIZ, Systematic Innovation and Technical Creativity* (trans. L. Shulyak & S. Rodman). Worcester, MA: Technical Innovation Center https://search.worldcat.org/title/The-innovation-algorithm-:-TRIZ-systematic-innovation-and-technical-creativity/oclc/42591715 (Russian original, 1973; in French, by the same author: *Et soudain apparut l'inventeur*, trans. A. Seredinski, 2006 https://www.decitre.fr/livres/et-soudain-apparut-l-inventeur-9782952139410.html). The critical assessment: Ilevbare, I. M., Probert, D. & Phaal, R. (2013). "A Review of TRIZ, and its Benefits and Challenges in Practice". *Technovation*, 33, 30-37 https://doi.org/10.1016/j.technovation.2012.11.003 ; Chechurin, L. & Borgianni, Y. (2016). "Understanding TRIZ through the Review of Top Cited Publications". *Computers in Industry*, 82, 119-134 https://doi.org/10.1016/j.compind.2016.06.002 . ↩ ↩

Part III — The machine

10. Beneath the interface, dragons

Old maps marked unexplored regions with a formula: *hic sunt dracones*, here be dragons. That is exactly the status of a software's engine in most interface discussions. People reason about what they see (the buttons, the panels, the sliders), and beneath that surface, for them, there are dragons: they know nothing about it, they see nothing there, so they don't talk about it. Anyone who sees, thinks, and cares only about the interface is a dangerous idiot, design or user-experience degree included. Incapable of understanding that an organism lives underneath, an engine (what the trade's jargon calls the *backend*, the software's back shop; both words will be used interchangeably here), he does not see that the interface is only the point of contact with that organism. This part of the book descends among the dragons, and this first chapter states its entry theorem: the complexity of an interface is not a property of the interface.

The right image is the dashboard, because a dashboard doesn't decorate: it drives. Old cars carried a choke knob, which enriched the fuel mixture on cold starts; it wasn't there out of a taste for knobs, but because the carbureted engine could not adjust its own mixture, so someone had to do it in its place — the driver was an organ of the engine. No dashboard stylist could remove

that knob: hiding the control would not have removed the function, only left the mixture with no one to adjust it. The choke did not disappear because someone tucked it into a menu; it disappeared the day electronic fuel injection made the engine capable of metering its own mixture — the control died at the surface when the function moved down into the machine, and the clutch pedal followed the same path with the automatic gearbox. The parameters and controls of a software interface are exactly that: not ornaments, but the controls of a machine running underneath. If the engine calls for 6 or 10 parameters, you cannot just hide them and live in denial, because someone will have to give them a value. Removing an element from the interface presupposes having first removed it from the engine, or having made the engine able to do without it, or to set it itself — injection, not a knob cover. Hence the theorem, and it must be taken literally: **when the interface is complex, it is already too late.** Its complexity is inherited, not created. The graphical interface is not parallel to the engine's architecture, it is its *termination*: the last segment, the one that breaks the surface. You don't solve an interface problem with mockups; you solve it by solving the engine problem the interface is a symptom of.

This theorem has a theoretical pedigree. Larry Tesler, one of the fathers of modern interaction, formulated what is known as the law of conservation of complexity: for a given task, there is an irreducible amount of complexity that cannot be removed, only moved; the only real question is *who absorbs it*, the user or the system[1]. Every parameter the engine cannot determine on its own is a share of complexity delegated to the user; every decision

the engine takes correctly in the user's place is a share absorbed. Absorbing, in someone's place, a difficulty they would otherwise have carried alone: the phrasing is an engineer's, the thing is not. Don Norman drew the overall lesson: complexity by itself is neither good nor bad; it is *confusion* that is bad[2]. A professional tool is legitimately complex, as the trade is; the design failure is complexity without structure, placed in the wrong spot, experienced as confusion. My contribution to this lineage is causal, and experience taught it to me the hard way: complexity *flows* in one direction, from the backend toward the interface. You therefore cannot simplify downstream without simplifying upstream; any simplification campaign that starts with the pixels ends up papering over the mess.

First case: in Darktable, hovering over a thumbnail in the lighttable triggered *18 queries* to the database, on every hover, to refresh metadata that had not changed since the previous hover. The symptom, on the interface side, is a sluggish photo library that drains batteries at the slightest cursor movement; the cause has nothing to do with the interface: the engine did not track the life cycle of its data, so it could cache nothing, so it refreshed everything, all the time. This is the case from the chapter "The root and the symptom", seen from the other end. Second case: the arbitration between explicit selection and hover selection (the ambiguity that overwrote metadata in the chapter "Measuring without lying to yourself") was "handled" by a file of about 1,100 lines, where twenty or so suffice once the ambiguity is settled in the engine: 1,000 lines of code to avoid making a design decision and to cover every possible case and user preference, some

wanting the hovered image included in their selection, and others not. Third case: the keyboard-shortcut system, overhauled to make *every* interface action configurable, including from MIDI controllers and game controllers. Result on the backend side: from 1,500 to over 4,000 lines, measured complexity up by a quarter; result on the user side: a list of several thousand configurable keyboard shortcuts in which it has become difficult to find the 3 shortcuts you actually need. The complexity added to the engine flowed all the way to the screen, as always, and the list of objects to configure reached an unmanageable length. Fourth case, in the opposite direction, because the theorem also predicts the remedy: Bauhaus, Darktable's in-house library of graphical widgets, the one that draws the sliders and the drop-down choices (*combobox*). Symptom on the interface side: spurious redraws and slowness in dispatching value changes. The widget waited for the image recomputation to finish before redrawing; the user, with no visual confirmation that his value had registered, repeated his gesture, which triggered a new recomputation without updating the visual confirmation any further, and the machine froze for long minutes. On the code side: the fourth most complex file in the whole project, where a simplification I had made in 2019 had been erased by feature inflation, including the MIDI support of the third case. The rewrite took me 3 weeks full-time, for a simple principle: redraw immediately, pass the value to the compute engine only at the end of the gesture. Cyclomatic complexity cut by a third, cognitive complexity nearly halved; and the interface, without changing its drawing, became responsive and predictable. We didn't simplify it: we simplified what it inherits.

A piece of software is an information system. What actually exists beneath the decor is data: text, the image, its metadata, the editing history, the settings, the caches. This data circulates through a flow that processes it, and forms the software's life cycle. Designing the architecture means deciding how that information circulates: which data structures exist, which module owns them, who is allowed to read and write them, through which interfaces the modules talk to each other, and how the software's core orchestrates the whole — who gets notified when a piece of data changes, in what order, and who arbitrates in case of conflict. That is the real object of software design work. And that is also what the user interacts with through the interface: a slider does not "do" anything, it writes a value into a backend parameter; a display does not "show" anything, it reads a state of the information system, which it converts into screen pixels. The graphical interface is only one client of the information system among other possible ones (the command line and scripts are others), and its quality is bounded by the quality of the data flow it consumes. Reread the 4 cases in that light. The 18 queries per hover: an interface forced to re-ask the database for the state of the data because nobody tells it what has changed. The 1,100-line file: one piece of data, the selection, without a single owner. Bauhaus's spurious redraws: a badly sequenced flow between the gesture, the display, and the render computation. None of these problems is visible on a mockup, and all of them show on screen — because the screen only ever shows the final state of the information flow.

How, then, does an information system absorb complexity without propagating it? The canonical answer dates from 1972 and has not aged: David Parnas showed that decomposing a system into modules means deciding which design decisions each module *hides* from the others[3]. A well-designed module locks a hard decision (a format, an algorithm, an execution order) behind a boundary the rest of the system cannot cross: the decision can change without anything else moving, a change inside the module can have no effect outside its boundaries, by construction. It is the same move as centralizing the masks in the chapter "Measuring without lying to yourself", one decision, one place; and it is what builds the engines whose interfaces can be simple: every decision the architecture locks away is a parameter the user will never see. The converse is just as true, and the cases above show it. A backend that does not decide (two competing selections, an indeterminate data life cycle) exports its indecision to the interface, where it becomes options, inconsistencies, and accidents. Architects sometimes say that a system's skeleton always ends up showing; for the interface, it is a law: *the screen is the X-ray of the architecture.*

The objection facing this chapter is a professional orthodoxy: designing "experience first," that is, drawing the ideal interface, then summoning the implementation to follow. It must be granted its share of truth, which is ours: yes, everything starts from the user — but from his *problem* (chapter "Stating the problem"), not from the pixels. The error begins when the drawing precedes the architecture of the information system: the drawing then fixes promises of data circulation that the backend

cannot keep, and you get unworkable designs, in conflict with what the software needs in order to function; then, with the deadline's help, the promised interface gets grafted onto the existing engine, and the graft produces exactly the quantified monsters above. The uncomfortable truth is that you need people who understand both floors, the user's trade and the engine's architecture, and they are rare, therefore expensive. But there is no alternative: the two floors are one. The most stripped-down dashboard in the world only lost its choke knobs because its engine learned to do without them.

If an interface's simplicity is won in the engine, it remains to be seen what simplicity is worth, what it costs, and why so many programs choose, checkbox after checkbox, option after option, exactly the opposite. That is the subject of the next chapter.

---

1. "Tesler's law", formulated around the early 1980s, has no primary publication; the citable source is Tesler's interview with Dan Saffer: Saffer, D. (2006). *Designing for Interaction.* Berkeley: New Riders https://openlibrary.org/works/OL9221137W . The popular attribution "in the mid-1980s at Xerox PARC" is chronologically dubious (Tesler had left PARC in 1980); the law itself is indeed his. Norman (see next note) credits it by name to Tesler, then at Apple, in chapter 2 of his book. ↩

2. Norman, D. A. (2010). *Living with Complexity.* Cambridge, MA: MIT Press https://openlibrary.org/works/OL15507110W . Verbatim:

"Modern technology can be complex, but complexity by itself is neither good nor bad: it is confusion that is bad." ↩

3. Parnas, D. L. (1972). "On the Criteria To Be Used in Decomposing Systems into Modules". *Communications of the ACM*, 15(12), 1053-1058 https://doi.org/10.1145/361598.361623 — the principle known as *information hiding*: each module hides a design decision that is likely to change. ↩

11. In praise of the simple

Pick up an old, fully mechanical film SLR. Three very general controls: the focus ring, the aperture ring, the shutter-speed dial. With that, you accomplish what digital bodies do today with the help of dozens of specialized submenus. Digital made the camera faster and more reliable in some contexts, but it added a considerable attention load to drive, at bottom, the same 4 parameters as in 1960: speed, aperture, (sensitivity), focus. The lesson of the object fits in one sentence, and it carries this whole chapter: **more options is not more control.** Control comes from well-chosen general controls; options, past a threshold, no longer add up, they bury one another. And this fact says absolutely nothing about the tool's efficiency or ease of use: we will see in this chapter that easy to use is not synonymous with simple.

Hiding a tree in a forest is the best way to make sure no one ever finds it again, and experimental psychology measured the phenomenon as early as the 1950s: decision time grows with the logarithm of the number of alternatives[1]. This is not a tax paid once, at learning time; it is paid at every use, by all users, including those who never asked for the option. The list of several thousand configurable shortcuts from the previous chapter is the illustration: every added entry makes the 3 useful ones harder to find. People sometimes invoke here the "paradox of choice" and its famous jam-jar study; the experimental truth is

more cautious, and the argument does not need it[2]: Hick's logarithm, combinatorial explosion, and maintenance costs are amply sufficient.

Combinatorial explosion is the second cost, and it is paid in the backend. Options do not add, they multiply. In Darktable, rendering a simple image thumbnail had to cope with 5 views and 7 display variants, plus the filmstrip: 36 combinations, multiplied by 3 graphical themes, or 108 sets of CSS style instructions to fully dress the interface. During a major graphical overhaul, several of those sets were simply forgotten; how could it have been otherwise? The Darktable team had implicitly committed — without knowing it — to maintaining and testing 108 style combinations across all the graphical widgets of the lighttable, when the whole thing looked like nothing more than an innocent effort to accommodate everyone, each with his own variant. A simple calculation would have sufficed to cancel the project on the grounds of the outsized scale of the task (for a project which, remember, declined to provide precompiled software packages on the grounds of the workload that represents), had the maintenance cost been part of the checklist of the chapter "Tooling up without submitting", and had the combinatorial analysis of the option surface to maintain been among the team's skills. No human tests 108 style combinations across the entire interface, so the untestable state space produces what it always produces: irreproducible bugs, dependent on a configuration the reporter doesn't even think to mention. And there is a third cost, more perverse: the option that lets the user harm himself. The same software acquired a fully configurable

module-layout tool, without a best-practices guide, which saved the layout under the *translated* names of the modules: changing the interface language was enough to lose your settings. Offering that latitude is not a service; it is letting the user decide how to hurt himself, and leaving him responsible for it. Still, denying it to him can be worse yet, and the objection has a name — paternalism: deciding in someone else's place, for his own good. The chapter "Enabling without holding the brush" will draw the line, and the criterion will be the same from one end of this book to the other: protecting someone means making him able to decide, never deciding for him, while still making the dangerous option a little more costly than the other, to make sure it is really what he wants. That is the principle of the override control.

A tool is not the sum of its features. The Swiss Army knife is the proof by absurdity: bristling with functions, and yet no craftsman has replaced his toolbox with a Swiss Army knife, because it is barely ergonomic, fragile, and you cannot tighten a screw to a proper torque with a flat 5-centimeter handle. Useful when camping; out of the question for work. The workshop provides the complementary image: when the benches, shelves, and tables are covered with ultra-specialized tools, the marginal productivity of each additional tool turns negative, because the time the tool saves is lost searching for it among the others, and Murphy's law dictates it is always at the bottom of the drawer or the box. Better a controlled number of tools, general and complementary, and tables reserved for work. Past functional completeness, adding more turns the tool into a toy: an object full of exciting functions that does not make you more productive,

but wastes your time searching for what you need among the clutter. A tool develops until it fulfills its purpose; after that, developing further is no longer improving. Development damages stability and breeds maintenance: it is literally work that creates work. A mature project must slow down, and that is not an admission of weakness, it is a sign of completion.

This economy has practical rules. Each feature is weighed against its lifetime maintenance cost, not against its appeal at the moment of writing it; the principle has a name among practitioners, YAGNI, *you aren't gonna need it*: implement when the need is there, never because it might come[3]. Priority is set by the usage data of the chapter "Counting the absent", and the negative example is already in this book: doubling the complexity of the shortcut system to serve MIDI controllers, owned by one user in 10 and used by two in 100, is priorities turned upside down. Finally, the hygiene rule I have applied since day one of my fork, when 22,000 lines were removed in one stroke: what works halfway is removed entirely. A half-working feature costs the price of a whole feature in maintenance, returns a fraction of the service, and manufactures continuous disappointment in the user. Removing code and options: robustness comes at that price.

In praise of the simple is not in praise of the simplistic. But the real misunderstanding runs deeper: when people demand *simplicity*, they almost always mean *ease*, and they believe the two go together. They are opposites. A violin is the simplest object in the world: 4 strings, a bow, 4 pegs, a shape unchanged for 4 centuries. There is nothing easy about it; it takes years to draw a true sound from it. The simple is the stripped-down,

general tool, the one the hand must adapt and train to. Its difficulty belongs to the trade, not to the object.

The easy is the reverse, and the mechanism is counterintuitive: you do not make a tool easy by simplifying it, you make it easy by *specializing* it. In the world of mechanical machines, specializing means removing possible uses and narrowing the field of use, to optimize the machine and its interface for one intended use (or a few). In the flexible world of computing machines, ease is bought by adding modes, presets, functions pre-cut for anticipated usage scenarios — so that the user has nothing to learn, the device having already decided for him. Each anticipated scenario is one more specialized machine. And since software stacks specialized machines with no physical limit, they get multiplied to cover the whole range of uses, until the interface is saturated with them and becomes unreadable. Ease pursued thus produces, precisely, the clutter the first half of this chapter measured. That is why the demand for "simplicity" so often ends in its opposite: what is being demanded under that name is ease, and ease is paid for in anti-simplicity.

Simplifying a work tool is therefore not making it easy. It is reducing the number of steps and the cognitive load *for the same task accomplished*, without ever dispensing with the trade. The difficulty must stay with the trade; it must never move into the tool. And the mirror image of the same trap exists: the applications with the most "simple"-looking surface often obtain that facade by degrading what does not show, starting with their color models. Everything has a cost; Tesler's law from the previous chapter takes no vacations. The chapter "The instrument

and the machine" will develop this opposition — the instrument versus the machine — for creative tools.

The objection facing this has the best reputation there is: flexibility as freedom. Who am I to decide in the user's place? Let everyone configure their tool to their own workflow; options respect the diversity of practices. This objection draws its strength from well-documented biases. Removing a function is experienced as a *loss*, and economic psychology has established that a loss weighs more than an equivalent gain: loss aversion, compounded by the status quo bias that makes people prefer the present state for the sole reason that it is present[4]. Add the weight of invested effort: the more time you have spent configuring a tool, the more you overvalue it — the "IKEA effect," where the labor supplied breeds attachment, regardless of the result[5]. The user then assimilates the options he has accumulated into his workflow, and removing a function is experienced as the amputation of a practice — hence the fear, recurrent in free software, that a simplification will "break someone's workflow." But a preference is not a workflow, and a pile of options builds none. Marketing has measured this under the name *feature fatigue*: before the purchase, the customer picks the product richest in features; in use, that abundance degrades his satisfaction, the acquired capability being paid for in suffered complexity[6]. Options orient no one, least of all the beginner, who needs to be guided, not put in front of a dashboard. Three answers, then. First, each person's right ends where everyone's tax begins: Hick's logarithm is paid by all users, at every use, including those who wanted to configure nothing; one person's

option clutters another's screen. Second, the actual track record of options added in the name of freedom is there for inspection: 108 style sets of which half go untested, settings lost on a language change, thousands of entries in the configuration file. Third and above all, the confusion is about the word: real flexibility is *generality*, not configurability. The generic tool — the aperture ring, the tone equalizer — serves uses its designer never anticipated, precisely because it exposes a small number of fundamental degrees of freedom instead of multiplying preconfigured special cases. The right answer to the diversity of practices is a general tool that bends to everything, not a configurable tool that unfolds into 108 states. The option retains a residual legitimacy, and the chapter "Stating the problem" gave the criterion: it is justified when it serves incompatible *needs* — accessibility, screen sizes, keyboard-only and mouse-only — never when it arbitrates preferences.

Do less, better: the program fits in 3 words, and it has a structural enemy — code itself, which accumulates far more readily than it retreats, and every line of which is a debt whose interest must be paid. That is the subject of the next chapter.

---

1. Hick, W. E. (1952). "On the rate of gain of information". *Quarterly Journal of Experimental Psychology*, 4(1), 11-26 https://doi.org/10.1080/17470215208416600 ; Hyman, R. (1953). "Stimulus information as a determinant of reaction time". *Journal of Experimental Psychology*, 45(3), 188-196 https://doi.org/10.1037/h0056940 . Choice reaction time

grows as $\log_2(n)$ of the number of alternatives (the Hick-Hyman law). ↩

2. The jam study: Iyengar, S. S. & Lepper, M. R. (2000). "When Choice is Demotivating: Can One Desire Too Much of a Good Thing?". *Journal of Personality and Social Psychology*, 79(6), 995-1006 https://doi.org/10.1037/0022-3514.79.6.995 ; popularized by Schwartz, B. (2004). *The Paradox of Choice*. New York: Ecco https://openlibrary.org/books/OL26343625M/The_Paradox_of_Choice . But the meta-analysis by Scheibehenne, B., Greifeneder, R. & Todd, P. M. (2010), "Can There Ever Be Too Many Options?", *Journal of Consumer Research*, 37(3), 409-425 https://doi.org/10.1086/651235 , finds a mean effect close to zero: choice overload only appears under certain conditions. The chapter's argument therefore does not rest on it. ↩

3. Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Reading, MA: Addison-Wesley https://books.google.com/books/about/Extreme_Programming_Explained.html?id=G8EL4H4vf7UC — the rule "You Aren't Gonna Need It": implement only what is provably needed, the anticipated feature being almost always badly anticipated. ↩

4. Kahneman, D., Knetsch, J. L. & Thaler, R. H. (1991). "Anomalies: The Endowment Effect, Loss Aversion, and Status Quo Bias". *Journal of Economic Perspectives*, 5(1), 193-206 https://doi.org/10.1257/jep.5.1.193 — we overvalue what we already own (the endowment effect); a loss weighs more

than an equal gain (loss aversion); we prefer the state in place for its own sake (status quo bias). ↩

5. Norton, M. I., Mochon, D. & Ariely, D. (2012). "The IKEA Effect: When Labor Leads to Love". *Journal of Consumer Psychology*, 22(3), 453-460 https://doi.org/10.1016/j.jcps.2011.08.002 — the effort invested in an object increases the value ascribed to it, regardless of its actual quality. ↩

6. Thompson, D. V., Hamilton, R. W. & Rust, R. T. (2005). "Feature Fatigue: When Product Capabilities Become Too Much of a Good Thing". *Journal of Marketing Research*, 42(4), 431-442 https://doi.org/10.1509/jmkr.2005.42.4.431 — before purchase, customers favor the product richest in features; in use, that abundance degrades satisfaction and usability, the number of capabilities turning against the product. ↩

12. Code is a liability

There is an accounting error at the heart of software culture: booking code as an asset. We measure work by the volume of code added, we celebrate big contributions, we mistake the bustle of repositories for health. It is the exact reverse of the truth. Code belongs on the liability side: every line written is a commitment to future spending, because it will have to be read, understood, tested, adapted and repaired by humans for the entire life of the software. The accounting metaphor is not mine: it is Ward Cunningham's technical debt, coined to explain that a shortcut taken in the code bears interest until it is repaid[1]. And the interest runs even without a shortcut: Lehman's laws, established empirically as early as the 1970s, say that a program in use must evolve constantly or die, and that its complexity grows mechanically as it evolves, barring explicit work to reduce it[2]. Code is a garden: every season brings its share of chores. Building a car is not blindly piling steel onto 4 wheels; at some point, the car has to drive. Success is measured in user problems solved, never in tonnage of code produced.

From this accounting follows the definition of good code, and there is nothing aesthetic about it: good code is code that is easy to maintain. You don't write code for computers; the compiler will swallow anything. You write code for the humans who will have to read it, understand it and repair it, and any fool can write code that a machine understands — writing code that a human

understands is the trade[3]. Kernighan and Plauger gave the most useful corollary: debugging code is twice as hard as writing it; if you write the cleverest code you are capable of, you are by definition not clever enough to debug it[4]. Boring is a virtue: the boring, predictable, guileless design is the best design. Maintenance, for its part, is the opposite of prestige: invisible, never applauded, relegated to the bottom of release notes, while users judge a project's health by its bustle. You gain nothing by maintaining; you lose everything by not maintaining; you pay either way. The verb, by the way, is badly housed on the chore side: the philosophers who thought about care made it the first word of their definition — to maintain, continue, and repair a world so that we can live in it. That is exactly the work this chapter describes, and exactly the work the milieu despises. Wisdom therefore consists in making maintenance first possible, then easy, and 3 rules almost suffice:

1. Write the least code possible: hunting a bug in 300 lines is a different job from hunting it in 3,000.
2. Write the simplest code possible: few steps, few assumptions, few special cases — which means, in practice, avoiding the options, variants and branches that the previous chapter put on trial.
3. Write code that is closed on itself (self-contained): isolated modules that communicate through the core of the system, so that whatever changes inside a module breaks nothing outside — that is Parnas, again, and it is the only one of the 3 rules that survives the software's growth.

A fourth rule wraps the other 3: think before you write. The quantity of code produced is inversely proportional to the quantity of thought invested, and you never extend a program you have not first managed to simplify.

Thinking before writing has a precise form in engineering: finding the right mathematical object. In 2011, Darktable's founder had settled the order of modules in the processing pipeline by the canonical method: developers declare local constraints on module ordering ("A before C," "B before A"), a script turns them into a directed graph, and a topological sort, an algorithm settled since the 1960s, derives the global order by graph resolution. Clean, minimal, correct. In 2019, users had to be allowed to reorder modules at run time. The feature was implemented by giving each module a numeric priority, a floating-point number, the pipeline order being that of increasing priorities. To insert a module between two others, you assign it an intermediate number: between 1.0 and 2.0 you place 1.5; between 1.0 and 1.5 you place 1.25, and so on (bisection). The scheme looks clever and it is wrong twice over. First, repeated insertions squeeze the numbers together until they exhaust the machine's floating-point precision, which is finite, and two priorities become indistinguishable, creating unsolvable errors. Second and above all, it encodes an *absolute position* where the problem is made of *relations*: what you know about a module is that it must come before or after some other module, not that it lives at number 1.25. The whole thing was grafted onto history-handling code that had been neither refactored nor simplified beforehand. Result: a backend so fragile that the maintainer who

had accepted the change had to rewrite it entirely the following year with a structure that says what it means, the linked list, where each module points to the previous one and the next. As for the topological sort, the only thing capable of merging two pipelines whose modules differ, the pre-compilation script that performed it had been deleted in 2019 without ever being ported into the software for dynamic use: the system's mathematical capability had regressed. The moral fits in one line: the improvised heuristic is a debt, the right mathematical object is a repayment. A graph, a sort, a linked list are not theorists' refinements; they are the shortest and safest forms in which to represent the problem in order to solve it correctly, and the ones that make you write the least code. There remains the question of *when* to rewrite rather than repair, for the rewrite is itself a potential debt. The criterion is neither the code's age nor your distaste for it: it is a change of requirement. You repair as long as the need is the same and the defect is local; you rewrite when the need has changed to the point that the structure can no longer carry it — and then you rewrite *by increments*, wrapping the old and diverting the calls one by one toward the new, never in a single-block overhaul that replaces everything at once. Rebuilding from scratch a whole piece of software that works is the classic strategic blunder, the one that has sunk entire products: along with the code, you throw away all the knowledge accumulated in its corners, only to reintroduce the bugs it had finally fixed[5]. Repairing a proven foundation is almost always worth more than a fresh prototype. All of which sums up as follows: use the right data structures, and use the right algorithms on top of them. As of this writing, those right

algorithms have mostly been known and published for more than 50 years, and it is futile to reinvent them, which implies researching the state of the art before diving in: yet another "administrative" task before finally getting to build something, which does not plead in its favor in a context where one of the open-source developer's main motivations is the pleasure of building something in a weekend.

There remains the question of dependencies, for you never write everything yourself, and one mantra governs the choice: "don't reinvent the wheel." The phrase is ignorant. The wheel is reinvented constantly, and precisely for performance: the wooden cart wheel, the steel-spoked wheel, the double-wall aluminum rim, the lenticular carbon wheel of time trials have nothing in common except being round. Each reinvention served a need the previous wheel did not serve. In software, "not reinventing" means depending on a third-party library, and a dependency is not free code: it is a transfer of liability bundled with a new contract surface. Libraries change their interfaces, and the code that works today breaks tomorrow, at a time chosen by someone else. Some are Schrödinger libraries: the camera access that worked two years ago no longer works today, nothing in the release notes mentions it, and it is the application's developer whom the user will take for an incompetent, not the shared library's. Distributions compile with their own options: a raw file format disabled by the packager cost me, the developer, two hours of diagnosis; the ordinary user never stood a chance. And in high-performance computing, generic libraries reload the image pixel by pixel at every step, where in-house code fuses the

loops and vectorizes, combining several operations into a single memory access.

AI programming agents, built on LLMs, are changing the game on this front: it is now easy, fast and reliable to let an LLM extract from a shared library the 2 or 3 functions you need, adapting them directly to your own needs and your own data structures, and possibly optimizing them for your use case. You then win on all 3 counts: by reducing the tedium of re-implementing classic algorithms, by reducing exposure to the bugs present in third-party code, and by sheltering yourself from that code's future API changes.

My definition of happiness fits in a fraction: happiness is the inverse of the number of people you depend on. The reconciliation with the chapter "Beneath the interface, dragons" is clean: you reinvent the engine when you have requirements the generic library does not serve and you want to control the code's failure surface, but you never reinvent interaction conventions. The double-click was invented 40 years ago and users don't have to relearn the world to indulge you. As for solving dependency hell by shipping every application with its own private copy of all its libraries (Snap and Flatpak software packages), in containers running in the background, that is treating obesity with obesity. Beneath the mockery there is a value that must be stated: wasted processor cycles, bloating dependencies and idle consumption are not housekeeping details, they are engineering faults. Efficiency is the very object of the trade — doing much with little — and software that squanders the machine's energy squanders that of the grid feeding it. Frugality

is not an austerity imposed on the user; it is the mark of work done well, and its opposite, the mark of work that has stopped counting what it costs.

The objection that caps it all is the culture of speed: ship value fast, assemble libraries, don't linger — code has become cheap, AI makes it nearly free. The answer is in this chapter's case files. Reordering modules by numeric priorities was the fast option: shipped fast, rewritten entirely the following year — the time saved was amply lost repairing the negligence that had allowed the time saving in the first place. Assembling libraries you don't audit is fast too, until the day one of them stops working and you are the one the user takes for the incompetent. Speed confuses the cost of writing code, which was never the problem, with the cost of owning code, which always was. You must iterate fast — the chapter "Decide, then hold" defended it; but iterating fast on solutions is no license to pile up liability fast, for the speed of the next iterations is bought with the parsimony of the previous ones. The only asset of a piece of software is the problem solved and the architecture that solves it legibly. Everything else, every single line, sits on the liability side, and the interest is running.

The good machine is disciplined: simple on the surface if it is simple in depth, parsimonious in its code, sovereign over its dependencies. The 3 rules of this chapter, like every principle that has proved itself, have a flip side: they can be recited without being understood, and a rule recited without being understood always ends up producing the opposite of what it promised. That is the subject of the next chapter, before we can finally say what this machine should look like to the person who uses it.

1. Cunningham, W. (1992). “The WyCash Portfolio Management System”. OOPSLA ‘92 experience report, *ACM OOPS Messenger*, 4(2), 1993 https://doi.org/10.1145/157710.157715 — first formulation of the debt metaphor: shipping immature code is borrowing; “every minute spent on not-quite-right code counts as interest on that debt”. ↩

2. Lehman, M. M. (1980). “Programs, Life Cycles, and Laws of Software Evolution”. *Proceedings of the IEEE*, 68(9), 1060-1076 https://doi.org/10.1109/proc.1980.11805 — notably the first law (continuing change: a program in use must adapt or become progressively less useful) and the second (increasing complexity: barring dedicated work, structure degrades as the program evolves). ↩

3. Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Reading, MA: Addison-Wesley https://openlibrary.org/books/OL7407595M, p. 15: “Any fool can write code that a computer can understand. Good programmers write code that humans can understand.” ↩

4. Kernighan, B. W. & Plauger, P. J. (1978). *The Elements of Programming Style*, 2nd ed. New York: McGraw-Hill https://openlibrary.org/books/OL4716911M, p. 10: “Everyone knows that debugging is twice as hard as writing a program in the first place. So if you’re as clever as you can be when you write it, how will you ever debug it?” ↩

5. Spolsky, J. (2000). “Things You Should Never Do, Part I”. Joel on Software, April 6, 2000 https://www.joelonsoftware.com/2000/04/06/things-you-should-never-do-part-i/ — the full rewrite as “the single worst strategic mistake” (the Netscape example). The incremental method: Fowler, M. (2004). “StranglerFigApplication” (martinfowler.com) https://martinfowler.com/bliki/StranglerFigApplication.html — replacing a legacy application by strangling it from the edges, function by function, never in one block. ↩

13. The cargo cult

During the Pacific war, Melanesian islands saw airplanes land loaded with goods. After the war, the airplanes stopped coming; so some islanders built runways, put fires along the sides, built a controller's hut with a man inside, two pieces of wood on his head like headphones and bars of bamboo sticking out like antennas. The form was perfect, but no airplane landed. Richard Feynman turned this image into the name of a pathology of science: work that follows all the apparent rites of research — protocols, figures, publications — but lacks the essential thing, the honesty that seeks to refute itself[1]. Software picked up the image as early as the 1990s to name its own version of the disease: ritual code, structures copied because they have always been there, methods aped without understanding what they accomplish[2]. This chapter applies the diagnosis to good practices themselves. For the previous chapter gave rules — the least code, the simplest, the most self-contained — and this book would be incomplete if it did not show how each of those rules, recited without its reason, produces exactly what it was meant to prevent. A good practice is the compressed form of a past pain; whoever applies the form without knowing the mechanism is building bamboo antennas.

Let us start with the most venerated: *don't repeat yourself*, DRY. The rule is sound and its founding text precise: what must exist in only one copy is each piece of *knowledge* in the system — a

formula, a business rule, a format — not each run of lines that look alike[3]. The authors themselves spent the twentieth-anniversary edition complaining about the misreading: "many people took it to refer to code only: they thought that DRY means 'don't copy-and-paste lines of source'"[3]. The ritual, for its part, hunts visual resemblance: two blocks that look alike — quick, a common abstraction. But two similar pieces of code do not necessarily carry the same knowledge — they may resemble each other by accident, today, and diverge tomorrow. The forced abstraction welds them together; every new case then adds a parameter, a branch, a flag, until the "common" function is a labyrinth where no one understands any of the cases anymore. A respected practitioner of the object-oriented school gave this dead end its lasting name — the wrong abstraction — and its exit rule, which is an apparent sacrilege: "prefer duplication over the wrong abstraction" — dismantle the common function, reintroduce the copies, and wait until you understand[4]. The guardrail, in fact, existed from the very origin of the refactoring movement: the rule of three — you tolerate the second occurrence and only abstract at the third, when the real common structure has become visible[4].

I paid to learn that this sacrilege is wisdom itself, and the proof comes from my own worksite. In Darktable, the thumbnail grid — the lighttable — and the horizontal scrolling strip — the filmstrip — both displayed image thumbnails. Two similar displays: the DRY reflex fired, and one and the same widget served both views, *re-parented* from one to the other depending on context. On paper, zero duplication — elegant. In practice, the widget's state

— scroll position, thumbnail initialization, memory release, event lifetimes — now depended on the view where it had *last lived*. Scrolling that jumps, thumbnails destroyed at the wrong moment, unpredictable garbage collection; and the re-parenting was so expensive that every switch between lighttable and darkroom froze the interface for nearly a second. All of it was fixed patch after patch, which is to say never: each fix added one more special case to the monster. The real fix, in Ansel, was to stop being clever: two separate thumbnail tables, one per view. A little more explicit structure, far fewer accidental states — and the switch between views made instantaneous. The duplication we had avoided was 20 lines; the non-duplication cost years of bugs. DRY had not lied: its recitation lied, for the two views shared no knowledge, only an appearance.

The same trial applies to *clean code*, and it must be conducted with a straightedge, for the previous chapter defends readability as the fundamental economy of code. The movement's canon — tiny functions, systematic indirection, polymorphism everywhere — has been erected into a professional morality: "clean" or "dirty" code, clean or dirty developers[5]. But those rules have a domain of validity, and it has borders. A high-performance-computing practitioner redid the canon's own example — the areas of geometric shapes, chapter 6 of the founding book — removing the prescribed virtues one by one:

- without polymorphism, 1.5 times faster;
- with plain tables, 10 times;
- vectorized, 20 times and more[5].

It is a practitioner's critique, not peer-reviewed, and it does not condemn readability: it bounds its kingdom. In glue code — interfaces, configuration, orchestration — the object hierarchy and the small function reign by right: that code changes often and is read more than it runs. In a pixel loop traversed 60 times per second over 40 megapixels, every indirection is paid in millions of copies; there, the flat array and the fused loop are not dirt, they are the cleanliness of the domain. Reciting "small functions, many objects" in the middle of an image pipeline is applying drawing-room etiquette in an engine room — the form of cleanliness, in the exact spot where it soils. And the same lesson holds in reverse for manual optimization: the compiler optimizes all the better when the logic is simple, and code "cleverly" contorted for speed regularly ends up slower once compiled. In both directions the rule is the same: the measured profile decides, not the doctrine.

The *design patterns* — those 23 design schemas cataloged by the "Gang of Four" and recited ever since like a breviary — offer the most instructive case, because the critique comes from inside the trade's elite[6]. The catalog is useful as vocabulary: naming "observer" or "factory" what you were about to describe in three sentences. But Peter Norvig showed that 16 of the 23 patterns become invisible or trivial in more expressive programming languages — they describe not laws of design but *gaps in the language* that the programmer fills by hand with his own discipline[6]. Paul Graham drew the brutal conclusion: seeing patterns repeat in your code is seeing proof that you are writing by hand, over and over, what the machine should generate —

“When I see patterns in my programs, I consider it a sign of trouble”[6]. The cult inverts the reading: where the master sees a symptom, the ritual sees a virtue, and code is judged all the more professional for stacking patterns. Applying a pattern is neither good nor bad: it is a cost — a layer of indirection — traded against a real need for decoupling. Without the need, only the cost remains, worn with pride.

There remains the most institutionalized practice: code review. Its legend says it catches bugs, and the legend has been measured. At Microsoft, defects are reviewers’ first declared motivation — and only one-eighth of actual comments, far behind style improvements, comprehension and technical conversation[7]; at Google, the internal history is owned up to: review was instituted not to find bugs, but to force code to remain readable and maintainable by others, and the median latency of a review there is under 4 hours — a regime of transmission, not of inspection[7]. In other words, code review works, but not for the reason the ritual invokes: it is the mechanism by which a team transmits understanding and defends readability — the very definition of good code in the previous chapter. The cargo version of review nevertheless exists, and it is twofold: the rubber-stamp approval, where you sign off without reading to feed the metric; and its mirror image, the test-as-clearance that the chapter “The transparent tool” will recount, where a machine’s green light excuses you from finding out what it actually saw. In both cases, the form of verification replaces the verification.

And I was able to lay out the arithmetic of this substitution on the project this book studies, for it fits in 3 numbers. On one side,

human capacity, on which the review literature has converged for 50 years: a review that finds defects moves at 150-200 lines per hour; beyond 450 to 500 lines per hour, detection collapses; a useful session exceeds neither 400 lines nor the hour[8]. On the other, the flow: in Darktable, the right to merge narrowed down to a single man — pull requests were merged by some fifteen people in the early 2010s, five or six in 2019-2020, two or three since 2022, and a single developer has signed between 95 and 100% of them every year since 2019. In the peak year, 2020, he merged 1,549 pull requests. At the measured median size — 156 lines changed — that is 242,000 lines in one year, and it is a floor: at the measured mean size (508 lines), nearly 800,000. Reviewing the floor alone, at the rate at which defects are actually found, would take more than 3.5 hours a day, 365 days a year — on top of a full-time job and his own 600 annual commits. Even at the collapse rate, the one where you still look but no longer see, it would take an hour and a quarter every day, in perpetuity. The timestamps settle what the arithmetic leaves open: a quarter of the merges occur less than 10 minutes after the previous one, and the *average* merge alone exceeds, in volume to review, the budget of a maximal review session[9]. The conclusion describes an architecture, not a man. When all of a project's code passes through the evenings of a single person salaried elsewhere, review in the literature's sense is arithmetically impossible, and only the ritual remains: the stamp, the green test, trust. Sampling confirms it at the artifact level: of 21 recent merges — all through the same hands — every single one carries a formal approval, and 19 carry *no* line-by-line review comment at all. The approval exists; it is empty. The bottleneck is not a lack of virtue — it is an

organizational design, and it has the defects of a design that no one ever made.

The cargo cult is not a modern pathology: it is a very old regime, and the philosophy of technique has mapped it. Jacques Ellul opened his master-work by positing, after Mauss, that magic is a technique in the strictest sense of the word — "we were right," he would say again 30 years later, "to consider that magic was also a technique"[10] — and he described its apparatus: rites, formulas and procedures fixed once and for all, because the slightest variation in a word or a gesture would break the magical equilibrium — and why would anyone change a means held to be the most effective? Magic, he writes, endorsing Masson-Oursel's formula, is the first expression of technique: a mediation toward powers one does not control, aimed precisely at trying to control them through codified rituals, as material technique is a mediation toward matter. And he added the warning that concerns this chapter directly: even today, when studying the techniques that address themselves to men, we draw without knowing it on the great current of magical techniques[10]. Now development methods, processes, recited best practices are exactly that — techniques that address themselves to men; as soon as their reason is lost, they fall back into the original regime: the fixed rite, performed for its form, supposed to produce efficacy over a domain one does not control. Ellul even described the magical regime's defense mechanism, and you would swear he was describing our forums: when the rite fails, one blames the magician, never the technique. *You didn't do agile properly; you applied the pattern wrong; you weren't SOLID enough.* The ritual is

unfalsifiable by construction: its failures accuse the officiant, never the rite. There is the general law that the chapter "Stating the problem" announced, and it has a sequel: 20 years later, Ellul said where this regime leads when it wins — "it is not technique that enslaves us, but the sacred transferred to technique, which keeps us from having a critical function"[10]. A sacralized practice can no longer be criticized, only recited or betrayed; the chapter "Thinking against oneself" will meet this impossibility of critique again, at the scale of an entire milieu.

There is an old English rule for fences: never take down a fence before you know why it was put up[11]. The cargo cult calls for the symmetrical rule, which is the law of this chapter: never apply a rule without knowing why it was written. The two rules are one and the same — in both cases, the unit of practice is not the rule, it is its reason. That is why, in my project, the requirements imposed on contributions — functions that do only one thing, one entry and one exit, two levels of nesting at most, complex conditions stowed away in boolean functions with readable names, comments that say *why* and never *what* — are published with their reasons, and that is why the final rule of those requirements authorizes suspending each of the others: the day a rule contradicts its reason in a given case, the reason commands. A discipline that can no longer state its whys is no longer a discipline; it is a liturgy, and liturgies do not make airplanes land. This law holds in advance against the present book: it will end with a code of ethics, and a code of ethics is a dangerous object — it is the exact form a rule takes when it is

about to become a rite. Each of its articles will therefore be given with its reason, and none will ask to be recited.

And the next chapter arrives at just the right moment to test this law, for a new instrument has just upended every rule of the trade, and no established liturgy yet exists for it: generative artificial intelligence. Never will the "why" have been more necessary than before the tool whose rules no one yet knows.

---

1. Feynman, R. P. (1974). "Cargo Cult Science". *Engineering and Science*, 37(7), Caltech https://calteches.library.caltech.edu/51/2/CargoCult.htm — Caltech commencement address, June 1974; reprinted in *Surely You're Joking, Mr. Feynman!* New York: W. W. Norton, 1985 (French translation *Vous voulez rire, monsieur Feynman !*, Paris: InterÉditions, 1985; reissued by Odile Jacob). The transposition to software came later and is not Feynman's. ↩

2. The term "cargo-cult programming" entered the Jargon File in early 1991 (printed in Raymond, E. S., ed., *The New Hacker's Dictionary*. Cambridge, MA: MIT Press, 1991) https://mitpress.mit.edu/9780262680691/the-new-hackers-dictionary/ : "a style of (incompetent) programming dominated by ritual inclusion of code or program structures that serve no real purpose". At the scale of organizations: McConnell, S. (2000). "Cargo Cult Software Engineering". *IEEE Software*, 17(2), 11-13 https://stevemcconnell.com/wp-

content/uploads/2017/08/CargoCultSoftwareEngineering.pdf . ↩

3. Hunt, A. & Thomas, D. (1999). *The Pragmatic Programmer: From Journeyman to Master*. Reading, MA: Addison-Wesley — the DRY principle there targets duplication of *knowledge*, not of text. The authors' repentance, 20th-anniversary edition (Addison-Wesley, 2020) https://pragprog.com/titles/tpp20/the-pragmatic-programmer-20th-anniversary-edition/ : "In the first edition of this book we did a poor job of explaining just what we meant by *Don't Repeat Yourself*. Many people took it to refer to code only: they thought that DRY means 'don't copy-and-paste lines of source.'" ↩ ↩

4. Metz, S. (2016). "The Wrong Abstraction". sandimetz.com, January 20, 2016 https://sandimetz.com/blog/2016/1/20/the-wrong-abstraction — "prefer duplication over the wrong abstraction"; when the abstraction is wrong, "the fastest way forward is back". The rule of three: Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Reading, MA: Addison-Wesley https://martinfowler.com/books/refactoring.html , chap. 2, attributed to Don Roberts (French translation *Refactoring*, Paris: Dunod, 2019). ↩ ↩

5. Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ: Prentice Hall https://openlibrary.org/books/OL16900934M/Clean_code (French translation *Coder proprement*, Pearson France). The

critique: Muratori, C. (2023). "'Clean' Code, Horrible Performance". computerenhance.com, February 28, 2023 https://www.computerenhance.com/p/clean-code-horrible-performance — a practitioner's critique, not peer-reviewed, on the shapes example of chapter 6 of *Clean Code*; measured factors from about 1.5× (without polymorphism) to 20-25× (tables and vectorization). The later exchange between Martin and Muratori is public (https://github.com/unclebob/cmuratori-discussion). ↩ ↩

6. Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley https://www.informit.com/store/design-patterns-elements-of-reusable-object-oriented-9780201633610 (French translation *Design Patterns : catalogue de modèles de conception réutilisables*, Paris: Vuibert). The internal critique: Norvig, P. (1996). *Design Patterns in Dynamic Programming*, Object World conference, May 5, 1996 (norvig.com) https://norvig.com/design-patterns/ — "16 of 23 patterns have qualitatively simpler implementation in Lisp or Dylan than in C++ for at least some uses of each pattern"; Graham, P. (2002). "Revenge of the Nerds". paulgraham.com https://paulgraham.com/icad.html — "When I see patterns in my programs, I consider it a sign of trouble." ↩ ↩ ↩

7. Bacchelli, A. & Bird, C. (2013). "Expectations, Outcomes, and Challenges of Modern Code Review". *Proceedings of ICSE 2013*, 712-721 https://doi.org/10.1109/icse.2013.6606617 — finding defects is the number-one declared motivation;

defect comments account for one-eighth of the coded sample. Sadowski, C., Söderberg, E., Church, L., Sipko, M. & Bacchelli, A. (2018). “Modern Code Review: A Case Study at Google”. *Proceedings of ICSE-SEIP 2018*, 181-190 https://doi.org/10.1145/3183519.3183525 — “Reviewing was introduced at Google to ensure code readability and maintainability” (Finding 1); the founding impetus, per one of Google’s first employees, interviewed for the study, was “to force developers to write code that other developers could understand”; median latency under 4 hours. ↩ ↩

8. Fagan, M. E. (1976). “Design and Code Inspections to Reduce Errors in Program Development”. *IBM Systems Journal*, 15(3), 182-211 https://doi.org/10.1147/sj.153.0182 — team inspection at 130-150 lines per hour for system code (figures given as conservative), sessions capped at two hours. Kemerer, C. F. & Paulk, M. C. (2009). “The Impact of Design and Code Reviews on Software Quality”. *IEEE Transactions on Software Engineering*, 35(4), 534-550 https://doi.org/10.1109/tse.2009.27 — the 200 lines-per-hour rate validated empirically: below it, 56-66% of defects found; above it, 45-51%, in continuous decline at each speed step. Cohen, J. (2006). *Best Kept Secrets of Peer Code Review*. Austin, TX: Smart Bear https://search.worldcat.org/title/554483981 — the Cisco study (2,500 reviews): optimal detection under 300 lines/hour, collapse beyond 450-500, sessions of 100 to 300 lines and 30 to 60 minutes; a study published by a vendor of review tools, not peer-reviewed, and cited here with that label. ↩

9. Merged pull requests of the Darktable-org/Darktable repository, exhaustive API survey on August 1, 2026, `merged_by` field — the person who clicks the merge, whatever the mode (merge commit, rebase, squash): 10,821 pull requests merged since 2012; 9,316 since 2019, of which 9,136 — 98% — by the same developer, with an annual floor of 95% (2020-2021) and 100% to within one unit since 2023. Peak year, 2020: 1,549 merges, one every 5.5 hours, nights and Sundays included. Sizes measured on a sample of 100 first-parent diffs of merge commits (50 recent, 50 from 2023): median 156 lines, mean 508, 90th percentile 945, maximum 6,729. The timestamps of the merge-commit subset (2,286 merges, 2019-July 2026, spread over 1,275 days) show morning (7-9 a.m.), evening (5-7 p.m.) and night (10-11 p.m.) peaks, local time, and a quarter of the merges less than 10 minutes after the previous one. Qualitative validation: of 21 pull requests merged in 2026, all 21 were merged by the same developer (no self-merge), median latency 11 hours; 19 of 21 without a single line-by-line review comment. His own commits: 599 over the last 12 months. ↩

10. Ellul, J. (1954). *La Technique ou l'Enjeu du siècle*. Paris: Armand Colin (reissued Economica, 2008) https://archive.org/details/technique-ou-lenjeu-du-siecle-la-jacques-ellul, chap. I, section "Développement historique". The passages on magic are rendered here in indirect discourse: their text was checked against the English translation (*The Technological Society*, New York: Knopf, 1964, pp. 24-27 — "magic is a technique in the strictest sense

of the word", after Marcel Mauss; the ritual fixity; "the common reaction is to blame the magician rather than the technique"), not collated against the French edition. The only direct French quotation is from Ellul, J., "Technique et Économie" (1984), in *Encyclopédie philosophique universelle*. Paris: PUF, 1989 (our translation in the body). The phrase "first expression of technique" is Masson-Oursel's, which Ellul endorses. Ellul distinguishes two parallel currents — material technique and magical technique — that seldom mixed: the application to software methods is ours. The sacralization (French verified): Ellul, J. (1973). *Les Nouveaux Possédés*. Paris: Fayard (reissued Mille et une nuits, 2003, p. 316) https://www.fayard.fr/livre/les-nouveaux-possedes-9782842057824/ — « Ce n'est pas la technique qui nous asservit mais le sacré transféré à la technique, qui nous empêche d'avoir une fonction critique » (our translation in the body). The chapter "The instrument and the machine" will return to Ellul on the tool and the machine. ↩ ↩ ↩

11. Chesterton, G. K. (1929). *The Thing*, chap. "The Drift from Domesticity" https://catholiclibrary.org/library/view?docId=/Contemporary-EN/XCT.165.html&chunk.id=00000011 — "Chesterton's fence": do not remove a fence before you know why it was put up. ↩

14. Intelligence upstream

Nobody has confessed more bluntly where the programmer's trade stands in 2026 than Kent Beck. Beck is not just anybody: co-signer of the agile manifesto whose trajectory the chapter "Stating the problem" recounted, inventor of *extreme programming*, craftsman of the practices that governed software development for 20 years. Here is his assessment, in 2026: "nobody knows. It's not just that I don't know. It's nobody knows"[1]. Generative artificial intelligence has swept away the corpus of software-engineering practice accumulated over two decades — his own included — and the industry is back, he says, in an exploration phase: try everything, cheaply, in every direction, because no established rule survived the change of tool[1].

In that vacuum, two discourses hold the stage, and both must be read closely, because neither says what people make it say. The first is prophecy. Anthropic's CEO announced in early 2025 that AI would soon write 90% of the code; the figure gets quoted everywhere, never the sentence that immediately follows: "But the programmer still needs to specify [...] what are the conditions of what you're doing, [...] what is the overall app you're trying to make, what's the overall design decision"[2]. Reread the omitted sentence: specify conditions, aim at an application, settle design decisions — that is, word for word, what this book calls designing. Even in the mouth of automation's prophet, AI writes

code; it does not decide the tool. Design stays human, and that is the whole subject of this chapter. The second discourse is a practice, the one Andrej Karpathy christened *vibe coding*: "fully give in to the vibes, embrace exponentials, and forget that the code even exists" — accept everything the machine proposes without reading the diffs, until, again in his own words, "the code grows beyond my usual comprehension"[3]. Karpathy reserved this for throwaway weekend tinkering; the practice spread far beyond, and the vocabulary of 2025 records the result: the Collins dictionary crowned "vibe coding" while Merriam-Webster crowned "slop," the generated mush — the method and its product, consecrated the same year[3]. Beck described where that slope leads, and he gave it a name: the iceberg. What AI happily regenerates is the visible part of a system — the interface, the apparent logic. Below the waterline lives everything the old code did without saying so: the compliance rules, the edge cases, 20 years of lessons hardened into fixes. Throwing away the iceberg having looked only at its tip is running headlong into serious trouble[1] — and you recognize the chapter "Code is a liability": the cost of code was never writing it, but understanding it.

So this chapter is not one more prophecy — the field overflows with them. It is a chapter of method: where does this instrument make a designer better, and under what protocol?

First, the facts, because they are contradictory and the contradiction is instructive. In GitHub's controlled experiment, developers recruited to implement a standardized web server, from scratch, went 55.8% faster with the AI assistant[4]. In METR's randomized controlled trial, 16 seasoned open-source

developers, working on *their own* repositories — mature projects, over a million lines, 5 years of average tenure — were *slowed down by 19%* by AI across 246 real tasks… while estimating, after the fact, that it had sped them up by 20%[5]. The two figures must be read together. On virgin ground with a well-specified task, the tool pays; on mature code you know better than it does, the tool costs — and, in both cases, *perception measures nothing*: METR's developers believed themselves faster while they were slowing down, exactly the stated preference against the revealed preference of the chapter "Decide, then hold", applied this time to engineers. The conclusion is not that the tool is good or bad; it is that it has no value of its own — its value is that of the protocol that frames it, and the rest of this chapter describes the protocol.

My thesis fits in one sentence: artificial intelligence serves to create knowledge and understanding of the task — and it must stop there, *upstream*. Three uses, in order. The first is the state of the art. Surveying existing methods is the preliminary step of any research and development work, and the classical tooling has collapsed: general-purpose search engines reinterpret queries to the point of having become unusable for technical work, and academic search engines demand that you already know the exact keywords — the vocabulary problem of the chapter "Working for the work", applied to the researcher himself. Before, the commonly used strategy was a rational laziness: take the method already implemented in your software library, for want of 3 spare years to prototype the others. The large language model (LLM) changes the nature of the step: it points to candidate methods from a mere description of the problem, then — and this is the

reversal — *it implements and compares them all* on your test bench. The state of the art stops being a bibliography; it becomes a measurement campaign.

The second use follows from the first: sweeping the solution space. Numerical methods iterate on a solution until the result converges; assisted exploration opens an axis of iteration above that: iterating on *the methods themselves*. Provide objective error metrics against a ground truth — no impressions: numbers, chapter "Measuring without lying to yourself" — and the assistant tries signal representations, estimators, implementation variants, forms hypotheses about its own failures, tests them, corrects itself, and comes back with the full table[6]. One must gauge what this sweep is, and what it is not: it is the exact inverse of training a neural network. The network learns uninterpretable weights — a black box, biased like its sample; the assisted sweep "trains" nothing: it *selects*, measurements in hand, among explainable methods — and what remains at the end is a mathematical model a human understands, written down in black and white, reimplementable in 10 years without any AI. One accumulates opaque liability; the other banks knowledge.

The third use is the prototype, and one must spell out what it means for a developer of desktop software written in C — the worst possible language for this work, verbose, dangerous, without a net. In C, prototyping costs so much that solution sweeping shrank to the bare minimum: you implemented *one* idea, the first defensible one, and you lived with it. A laboratory could afford better; a lone man could not. The assistant gives the solitary developer back the economics of a laboratory: the Python

mock-up in an hour, the test bench by the afternoon, 15 variants measured and buried within the week[6]. I measured it on one dated workstream: the rewrite of the history system from the chapter "Code is a liability", restored topological sort included, took me 3 days instead of the 3 weeks I had estimated. And it swallows the chores that never deserved a brain: repetitive infrastructure code, and archaeology — reconstructing the call tree of a function through 300,000 lines, work that exists at all, the chapter "Code is a liability" said it, only because code is not self-contained. I drew from it a use I did not expect: having it read the assembly that the GCC compiler pulls out of my C. Checking what the compiler actually did — vectorized or not, fused or not — is more reliable than blind timing, which measures the machine's noise as much as the code; we are back to the chapter "Measuring without lying to yourself": read the artifact rather than guess from an aggregate number. All of this sums up in one stroke: use *high-tech* at research time to produce better *low-tech* at execution time.

These figures nevertheless suggest a man 10 times more productive, and that is not what happened. The fatigue changed object; it did not change volume. The fatigue of prototyping everything yourself and racking your brain over every semantic and idiomatic detail of the language you use has been replaced by the fatigue of reviewing the volume of code a machine produces in so little time — and if you hold to the only defensible rule, commit to the repository nothing but code you have read and understood, that review costs most of what the writing would have cost. It is the chapter "Code is a liability" found again from

the other end: the cost of code lies in reading and understanding, and that cost the assistant does not remove — it concentrates it on you. There, incidentally, lies a plausible share of the slowdown METR measured: the minutes saved not writing are lost again reviewing. So the real gain is not one of quantity; it is one of nature: attention leaves the implementation details — which will have to be reviewed anyway — and settles where nothing can be delegated: the architecture, the method, the criteria. You do not end the day less tired: the fatigue of doing has merely turned into the fatigue of reviewing.

That leaves the protocol, because speed without it is exactly *vibe coding*, and I published mine along with the mathematics it produced: Ansel's highlight reconstruction, rewritten in 2026 at the end of a human-machine campaign documented end to end[6]. The rules fit in a few lines. Every variant is implemented in the Python mock-up *and* in the C, compared across the two implementations, measured on every synthetic ground-truth scene and on natural images, tabulated — and nothing is retained without my explicit approval. A single autonomy is delegated: if a variant degrades *all* the metrics, the assistant may bury it alone without human review; it may never accept alone. These two rules were enough to sweep the solution space at machine speed without the design slipping away from me — and the division of labor that comes out of it deserves stating, because it is general: to the human, the physical intuitions, the criteria, and the eye — variants won on every metric and were rejected on images, because numbers do not see everything; to the machine, the error bars, the endurance, and that useful coldness which declares your

favorite hypothesis measurably wrong in the same even tone with which it buries its own. The instrument's vices are real: it has its pet methods and advises against the others — I had to force the exploration of avenues it disapproved of, and one of them won; when metrics contradict each other, it settles things "as best it can" by criteria that are not yours — hence the full-table rule, always; it forgets instructions, and it silently discards whatever it judges too expensive. It is not a colleague — and it is not a mere tool either, for a tool does not work while you cook, a tool takes no initiative, a tool has no opinion about the shape your code should take. It is an intern: it knows only what is taught in school, offers fresh ideas and absurdities with the same aplomb, demands unambiguous instructions and constant supervision — and its errors are devious, because they are buried in the middle of code that is otherwise perfectly valid. A prodigy of an intern, then, and a stubborn one; this chapter gives it a workstation and a rulebook. And the rulebook answers Beck's iceberg: if the submerged part stays understood, it is because here *nothing passes without being understood* — the protocol forbids precisely throwing away what has not been explained. As for the risk of unlearning the trade — systems nobody will understand anymore in 10 years, while a third of employers already say they are replacing part of their junior hires with AI[1] — the answer is the same: the campaign's deliverable is not the code, it is the article that fixes its mathematics, never written down anywhere before, the code's documentation, the comments, the flowchart. The knowledge is banked; the code is only its consequence. And the accounting lesson of the chapter "Code is a liability" comes out of the experiment hardened: AI drives the cost of *writing*

code toward zero, while the cost of code was never in the writing, but in the reading, the understanding and the maintenance — a tireless intern producing liability at high speed makes parsimony more vital, not less.

That leaves the price, payable in cash rather than in promises. Training and running these models costs dearly in energy, and the literature gives more nuanced orders of magnitude than the slogans: the famous "five cars" figure did not concern the training of one model but an entire architecture-search campaign, estimated rather than measured, and the estimate was later revised downward by a considerable factor; at the other end, the only large model with a fully published carbon footprint — trained in France, on a nuclear-powered grid — weighs some fifty tonnes of CO_2 over its full life cycle, and it is the electrical grid, more than the algorithm, that makes the difference; and generative inference, for its part, costs orders of magnitude more than classical processing, multiplied by billions of requests[7]. Does the low-tech produced downstream repay this upstream expense? Probably not, and I refuse to pretend otherwise — all the more so since an old paradox lies in wait: Jevons showed in 1865 that the more economical use of a resource *increases* its total consumption — against intuition, "the very contrary is the truth"[8]; efficiency made cheaper gets spent more: its extreme form is the rebound effect, which buries the fact that frugality can only ever be a discipline (hence a political choice), for it will never be a law of the market. What I defend is narrower and stands up: *the expense once made*, there are two directions. Embedding the high-tech in the product — inference in every

export, the black box in every home — condemns you to pay the cost at every run, at every user's, forever. Spending it upstream to design frugal, explainable, durable and repairable tools — the low-tech criterion as Bihouix stated it for any sustainable technique[9] — concentrates the cost where it is paid only once, and delivers to the user software that consumes *less* than before. It is the only form of the expense I know how to defend: the kind that goes out.

Hence my rule of conduct, and it wraps up the whole chapter: I use artificial intelligence assuming I will no longer have access to it tomorrow. Price, license, censorship, bankruptcy, regulation: access is a tap other people hold, and the chapter "Code is a liability" said what dependencies you do not control are worth. Everything the assistant produces must therefore outlive the assistant: written mathematics, test benches that run on their own, documented protocols, standard code without superfluous third-party layers — workflows and knowledge that last, usable without AI, by me or by someone else, in 10 years. Artificial intelligence is scaffolding: what you build with it must stand without it. The rest — the finished tool, what it owes the person who uses it — does not change one iota, and the answer still fits in one word that the next chapter will have to earn: transparent.

---

1. Beck, K., interview with G. Orosz, "How Kent Beck Shapes the Software Engineering Industry." *The Pragmatic Engineer Podcast*, July 1, 2026 https://newsletter.pragmaticengineer.com/p/how-kent-beck-shapes-the-software — "nobody knows. It's not just that I

don't know. It's nobody knows" (official transcript, at 2 h 14 min); "That playbook has been wiped clean" (at 2 h 18 min); the *vibe coding* "iceberg," with the example of a payroll computation whose regulatory compliance is the submerged part. The exploration phases: Beck, K. (2016). "The Product Development Triathlon" https://medium.com/@kentbeck_7670/the-product-development-triathlon-6464e2763c46 and "Fast/Slow in 3X: Explore/Expand/Extract" https://medium.com/@kentbeck_7670/fast-slow-in-3x-explore-expand-extract-6d4c94a7539 . Medium. The junior-hires figure: GMAC (2026). *Corporate Recruiters Survey 2026* https://www.gmac.com/-/media/files/gmac/research/employment-outlook/2026-corporate-recruiters-survey/report.pdf — one third of employers worldwide report having replaced at least part of their entry-level positions with artificial intelligence (621 recruiters, 39 countries; the sample covers the recruitment of business-school graduates). ↩ ↩ ↩ ↩

2. Amodei, D., "CEO Speaker Series." Council on Foreign Relations, March 10, 2025, official transcript https://www.cfr.org/event/ceo-speaker-series-dario-amodei-anthropic : "I think we'll be there in three to six months—where AI is writing 90 percent of the code. And then in twelve months, we may be in a world where AI is writing essentially all of the code. But the programmer still needs to specify, you know, what are—what are the conditions of what you're doing, what—you know, what is the overall app you're trying to make, what's the overall design decision?" ↩

3. Karpathy, A., post on X, February 2, 2025 https://x.com/karpathy/status/1886192184808149383: "There's a new kind of coding I call 'vibe coding', where you fully give in to the vibes, embrace exponentials, and forget that the code even exists"; further on: "I 'Accept All' always, I don't read the diffs anymore. [...] The code grows beyond my usual comprehension" — offered, he specifies, for throwaway weekend projects. "Vibe coding," 2025 word of the year of the Collins English Dictionary (November 6, 2025) https://blog.collinsdictionary.com/language-lovers/collins-word-of-the-year-2025-ai-meets-authenticity-as-society-shifts/; "slop," 2025 word of the year of Merriam-Webster. https://www.merriam-webster.com/wordplay/word-of-the-year ↩ ↩

4. Peng, S., Kalliamvakou, E., Cihon, P. & Demirer, M. (2023). "The Impact of AI on Developer Productivity: Evidence from GitHub Copilot." arXiv:2302.06590 https://arxiv.org/abs/2302.06590 — 95 developers recruited, standardized task (an HTTP server in JavaScript, from scratch), 55.8% faster with the assistant; non-peer-reviewed preprint, and heavy attrition (about one third of the recruits in the final denominator). ↩

5. Becker, J., Rush, N., Barnes, E. & Rein, D. (2025). "Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity." METR, arXiv:2507.09089 https://arxiv.org/abs/2507.09089 — randomized controlled trial, 16 developers, 246 real tasks on their own repositories (22,000 stars and one million lines on average, 5 years of

tenure): AI allowed *lengthens* completion time by 19%, where developers forecast a 24% gain and estimated a 20% gain after the fact. A snapshot of early 2025 (tools: Cursor with Claude 3.5 and 3.7), which METR itself presents as dated. ↩

6. Pierre, A. (2026). "The Mathematics of Guided Laplacians and Harmonic Transposition." ansel.photos, resources https://ansel.photos/en/resources/guided-laplacian-highlights/ — the section "How this work was actually done" documents the complete human-machine protocol (ground truth, dual compared implementations, mandatory human approval, bounded self-rejection rule, documentation as a bug detector) and the attribution of ideas between the author and the assistant; the section "Where LLM shines" details the automated state of the art, the sweeping of methods against objective metrics, and the assistant's biases. ↩ ↩ ↩

7. The "five cars" estimate: Strubell, E., Ganesh, A. & McCallum, A. (2019). "Energy and Policy Considerations for Deep Learning in NLP." *ACL 2019*, 3645-3650 https://doi.org/10.18653/v1/P19-1355 — the figure (626,155 pounds of CO_2) covers a complete neural architecture-search campaign, not the training of one model (training one Transformer weighs 192 pounds there); the estimate was recalculated downward by a factor of several dozen by Patterson, D. *et al.* (2021), arXiv:2104.10350 https://arxiv.org/abs/2104.10350. The measured footprint: Luccioni, A. S., Viguier, S. & Ligozat, A.-L. (2023). "Estimating the Carbon Footprint of BLOOM." *JMLR*, 24

https://www.jmlr.org/papers/v24/23-0069.html — 50.5 tonnes CO_2eq over the full life cycle, on the French grid at 57 g/kWh: it is the grid, more than the algorithm, that makes the difference with the estimates for other models. Inference: Luccioni, A. S., Jernite, Y. & Strubell, E. (2024). "Power Hungry Processing: Watts Driving the Cost of AI Deployment?" *ACM FAccT 2024* https://doi.org/10.1145/3630106.3658542 — generative tasks cost orders of magnitude more than discriminative processing. ↩

8. Jevons, W. S. (1865). *The Coal Question*. London: Macmillan https://www.econlib.org/library/YPDBooks/Jevons/jvnCQ.html, ch. VII, p. 103: "It is wholly a confusion of ideas to suppose that the economical use of fuel is equivalent to a diminished consumption. The very contrary is the truth." Modern treatment: Alcott, B. (2005). "Jevons' Paradox." *Ecological Economics*, 54(1), 9-21. https://doi.org/10.1016/j.ecolecon.2005.03.020 ↩

9. Bihouix, P. (2014). *L'Âge des low tech : vers une civilisation techniquement soutenable*. Paris: Seuil, coll. "Anthropocène" https://www.seuil.com/ouvrage/l-age-des-low-tech-philippe-bihouix/9782021160727 (revised ed., Points, 2021; English translation *The Age of Low Tech*, Bristol University Press, 2020) — faced with resource limits, sober, durable, repairable techniques of controlled complexity, rather than escalation. ↩

15. The transparent tool

What does the machine owe the person who uses it to create? This chapter's answer fits in one image that must be taken seriously: image-processing software is a virtual film lab. Surgery is nowadays performed remotely, a surgeon steering a robotic arm from a console; the interface of photo software is exactly that, the control desk of an arm that manipulates a simulated film. The film is not in another country but in another dimension: that of pure information, reconstructed from the sensor's data. The user is in the physical world. The software is the portal linking these two parallel worlds, and the remote control of real processes simulated by and within flows of information. From this metaphor everything else follows: the software's role is not to be intelligent, nor to make decisions in the artist's place like some virtual lab technician; it is to create axes of motion and degrees of freedom for a *human* technician operating at a distance. I call transparent the tool that honors this contract: the one that transmits the intention without adding its own opinion, as the brush transmits the gesture. Nobody asks a brush for its opinion, and that the brush is incapable of giving one is not a vice but a virtue.

The aesthetic consequence of this contract is the heart of the chapter: **aesthetics is achieved through fineness of control, not through the tool's taste.** And about the tool's taste, let us be clear: it is the taste of its designers, imprinted in the tool.

What matters to an artist is not the beauty of the out-of-the-box result; it is the distance separating the algorithm's output from the result he had in mind, and his ability to close that distance by pushing the controls. We are still inside the master idea of the first chapters: closing, at least partially, the gap between a current state and a preferred state. Serving aesthetics, for a tool, therefore means cutting the process into elementary steps and offering, at each step, the degrees of freedom needed to steer the trade-offs of an imperfect pipeline. The user drives; our job is to make sure the steering wheel turns fast enough and the gear stick is not too stiff. Each step does only one thing, with the fewest possible side effects, for non-orthogonal steps condemn the user to circular editing, where fixing the contrast detunes the color, which will have to be fixed again. And good out-of-the-box taste is not a design goal: "it looks good" is the responsibility of the entire pipeline, under the artist's supervision, never that of an isolated algorithm.

For the phrase that excuses everything, "it looks good," does not hold. Good for whom? Under what conditions, what settings, on what samples? Aesthetic preferences do not fall from the sky: they depend on *habitus*, that system of embodied tastes which sociology has shown to be socially constructed, where artistic education and the training of the eye weigh heavily[1]. The "good" of the researcher validating his algorithm is not that of the photographer who trained her gaze on 20 years of prints, and researchers have already declared "aesthetically pleasing" results that were unusable for a trained artist. The samples that illustrate publications are hand-picked; a stopped clock tells the right time

twice a day. The academic world of image processing lives off this merry-go-round, besides: find the edge cases of the star algorithm, publish an *ad hoc* algorithm that fixes them while creating its own, and get mocked by the next paper, which will do the same. No empirical test can be exhaustive enough to guarantee that a method "works" when "working" is defined in subjective taste. The way out of this merry-go-round is the discipline of the chapter "Stating the problem": an algorithm's quality is judged *solely* against its declared objective, its constraints and its success criteria, phrased as verifiable properties — no halos, chromaticity preservation, reconstruction fidelity. The decisive test of an expressive tool is the **result known in advance**: give a practitioner a rendering he knows and is aiming for; the specification decides whether the tool can reach it, and the trial verifies it. A tool is judged on its ability to deliver what the artist knew he wanted, not on what the tool proposes in its place.

These criteria are not wishes; they can be tested. A concrete example: when two editing tools act one after the other on an image, neither must introduce a stray tint where it touches none — a gradient that should stay neutral must not drift green or magenta at its edges. That can be made into an automatic test: run a synthetic image through the algorithm and check its output. Here design meets research: my blur and sharpness module was born from a physical model of diffusion — the heat equation pushed to a high order, first to imitate watercolor bleeds — then from the remark that flipping the sign of the diffusion term made the same model *deblur* instead of blurring. One single

mathematical structure, two opposite uses, obtained cleanly because the model had a physical meaning instead of being a cobbled-together empirical tweak. I had earlier attempted more ambitious deconvolution methods; I abandoned them, not for lack of interest, but because their compute times were unrealistic and their results unpredictable — a tool whose behavior cannot be guaranteed is not a transparent tool, whatever its theoretical elegance.

On what, then, should models rest, if taste is disqualified as a foundation? On physics, for it is the only ground truth available. Pixels are not anonymous data: they encode a light emission, the sensor records light, the screen emits it; it is a pipeline where light is both the incoming and the outgoing material. Digital imaging had a fit of hubris when it invented anonymous color transformations, anchored neither in the physics of photons nor in psychophysics, driven by unitless parameters disconnected from the film legacy. The result: digital processing built itself *beside* the analog heritage instead of on top of it, and the knowledge acquired on one side no longer transfers to the other, in either direction, even though cameras still have the same speed, aperture and sensitivity, and still record the same light. Meanwhile, computer graphics took the opposite path, modeling the physics of photons all the way to the spectral domain (and no longer just RGB), and today produces renderings more believable than digital photography when the latter is badly post-processed. Painters, for their part, mixed their pigments on purely physical grounds for millennia, without naming the chemistry they exploited, and trained their eye to predict the result. The

architecture lesson is that of model-view-controller: that the interaction model is physical in no way prevents the result from being a totally subjective perception — it only guarantees that what the artist manipulates behaves like the world his eye is used to. To which are added the previous chapter's requirements, signal-processing edition: continuous, monotonic, invertible transformations with physically plausible asymptotic behavior; and a property that deserves its name, graceful failure. We know in advance that parameters pushed to the maximum will produce an ugly result; judging visual quality under those conditions teaches nothing. What is evaluated is how the design constraints hold even under unreasonable settings: the algorithm that fails predictably and coherently can be improved; the one that fails at random is fit for the bin: you cannot improve randomness.

These criteria, however, are only worth anything if you *read* them, and this is where a test can turn into its own opposite. A test suite measured, on every change to the engine, the induced color deviation against a reference, with a tolerance threshold. One day, a fix was merged in my absence on a color module I had authored; the test reported a non-zero deviation. Yet the mathematical nature of the change demanded an *exactly zero* deviation on the test image: the non-zero number, however small, proved that an abnormal interaction had occurred elsewhere, and someone should have tried to understand. The change was accepted because the mean error on the image filter's output was under the threshold, and therefore presumably painless. That is the Goodhart's law of the chapter "Measuring without lying to yourself" caught in the act: the threshold, designed as an alarm,

served as a sign-off; nobody asked *why* a deviation expected at zero was not zero, since the deviation was below the alarm value. A validation criterion does not absolve — it alerts; mistaking it for a pass permit is turning the instrument of rigor into an exemption from rigor[2].

The contemporary objection is an entire school of thought, and it has the wind, the money and the processors on its side: full automation. Let the tool decide, since it "knows"; two-click magic for users who dream of complex software that spares them a complex skill; AI applying the learned style. The answer begins by deflating the vocabulary, and Aaron Hertzmann did it better than anyone: the best-performing machine-learning algorithms are *glorified data-fitting* — fitting curves of very high dimension to millions of points[3]. There is no taste in there; there is only the average of a training corpus, that is, the majority habitus of those who produced and labeled the data, applied to your image. Automatic correction is not neutral: it is someone else's opinion, force-fed and unsigned. This does not condemn automation: nobody regrets autofocus. It assigns it its contractual place: automation is admissible as a *proposal* — a starting point the artist can inspect, temper and refuse — never as a verdict. Hence the two non-negotiable clauses of the transparent tool: every automatic feature is paired with a switch that disengages it, and the algorithms' internal assumptions are kept to a minimum, documented, and modifiable at runtime — an exit door for the user, and incidentally for the designer, the day one of his assumptions proves wrong. The tool that decides in the artist's place and cannot be disengaged is no longer an instrument: it is a

vending machine for results, and the result it dispenses is average, in every sense of the word, by construction. Here we hold, in the form of a technical clause, a dividing line that applies far beyond images: deciding for someone is a confiscation, making him able to decide is its exact opposite — and nothing resembles the former more than an automatic feature designed, in perfect good faith, for its recipient's own good.

This chapter has defined the contract of expressive software: remote control, not intelligence; fineness of control, not factory-issued good taste; verifiable criteria, not "it looks good"; physics as the base, perception as the judge; graceful failure; universal disengagement. But software is only a recent case of a much older affair: what an instrument owes the artist, the luthier's craft has known for centuries, and the forced computerization of artistic practices erased it in one generation. That is the subject of the next chapter.

---

1. Bourdieu, P. (1979). *La Distinction : critique sociale du jugement*. Paris: Minuit https://catalogue.bnf.fr/ark:/12148/cb34628792q — taste as a socially acquired disposition. The concept of habitus is elaborated in Bourdieu, P. (1972). *Esquisse d'une théorie de la pratique*. Geneva: Droz https://doi.org/10.3917/droz.bourd.1972.01 . ↩

2. The color deviation is measured there in ΔE (CIELAB), the standard perceptual distance between two colors. The episode is public: Darktable repository, pull request no.

12233 https://github.com/Darktable-org/Darktable/pull/12233, August 2022 — a maximum deviation of about 1.6 ΔE accepted as under the threshold, where the transformation demanded 0, without the underlying anomaly being investigated. ↩

3. Hertzmann, A. (2018). "Can Computers Create Art?" *Arts*, 7(2), 18 https://doi.org/10.3390/arts7020018: "Today, the most-successful AI and machine learning algorithms are best thought of as glorified data-fitting procedures." ↩

16. The instrument and the machine

> *"An object should do as much as it can, and then disappear. [...] If an object is loved deeply, used habitually or sentimentally, or is given away as an heirloom, the object has succeeded in its highest form."* — Anonymous manifesto circulating online, source unknown.

I learned the piano, and I taught it. Every student knows the paradigm: *shut up and practice* — where "practice" is meant the way a musician works on a scale or a difficult passage: developing, through exercise and theory, the skill and the understanding of what you are doing. Nothing to do with the productivist injunction to wear yourself out on the job. Training to acquire and maintain agility, repeating the same bars until the sound is right, working the quality, the texture, and the expressiveness of the sound through the weight laid on the keys, the speed of the attack, the motion of the wrist and forearm. Using your body for fine motor control and training it beyond its current capacity, until perfectly artificial gestures become reflexes. And, in the end, learning to hear in your head the sound you are after before your hands even move. None of this will ever be reducible to sliders, variables, or algorithms. Which is why I have never understood the passion of so many photographers, often the most technophile ones, for push-button workflows. Everything happens as if they had been punished by digital

photography, sentenced to go through software to get their images, so the chore had to be reduced to a soulless procedure where everything that can be automated must be. People trade presets, some sell them, and artificial intelligence now applies someone else's style. Where is the joy in industrializing artistic creation? What is the point of a hobby that weighs like a chore? Art is a process, not a procedure. What sick mind would design a robot to do pottery, engraving, or flower arranging in your place? At what age do you lose the appetite for doing things yourself, even badly?

But the punishment is real all the same: photographers did not choose computing, it fell on them. In 20 years, digital replaced a 160-year-old technology, known and mastered, and in the operation the computer is pure overhead: a structural cost that produces nothing by itself. This conscription — the computer imposed for work that was done perfectly well without it — has a human cost that the chapter "Three-tiered violence" details. The film photographer was not necessarily a printer: he handed developing and printing to a lab. Digital put the development of raw negatives back on his shoulders, with no training in color management or in general computing — which pushed a generation toward software simplified down to a toy, which then *defined* the common expectation of what photo software should be. The darkroom printer worked with his hands in the finished product, paper and negative physically accessible, open to every hijacking (dodging, toning, solarization), including the ones the manufacturers had not planned; digital drove the artist out of the medium — the image lives in the computer as pure data, only

engineers know how to reach it, and the artist is confined to the functions the interface deigns to offer. And the language itself was written by computer scientists rather than color scientists, and less still by photographers: the previous chapter said what that cost. The sociology of work has documented the professional version of this forced conversion: from the late 1990s on, photojournalist hiring began requiring software skills, and the trade tipped toward the screen[1]. You then understand where the push-button appetite from the start of this chapter comes from, and you have to grant it this much justice: it is a reaction. Computerization first merged two trades by force, ordering the photographer to become his own printer as well, without training and often without taste for it; and this unwanted burden created the market for its opposite. The preset sells relief, in the form Braverman called the *separation of conception and execution*, the principle of Taylorized industrial work[2]: someone else designed the look, you apply it. Extending the concept to leisure is my own move, and it gives the measure of how strange the result is: to flee an imposed chore, a hobby ends up reproducing, at home and on free time, the factory's division of labor — people escaped forced printing by becoming the line worker executing someone else's taste.

The full measure of what has just been stated must be taken, for it is the heart of this whole book, and photography is only the particular case I know from the inside. What happened to photographers has happened to everyone, in every trade, right down to the counters of public administration: an entire world was artificialized without asking those who inhabit it, and every

day it manufactures forced users who did not choose to live in it and must live in it anyway. Someone has to help them do so. That is the thesis the conclusion of this book will defend.

The factory is no idle image; it is the historical background of the case. Lewis Mumford fixed the distinction: the tool is flexible — a knife smooths wood, carves it, splits it, forces a lock, drives a screw if need be — and lends itself to *manipulation*; the machine is specialized in a single operation and tends toward *automatic action*[3]. Jacques Ellul described what the reign of the tool presupposed: in traditional technique, the tool being insufficient by nature, it was the worker's skill that compensated — the emphasis fell on the human using the tool, not on the tool; and he described the modern machine, so specialized that at the first change of need it is worth nothing[4]. I would extend this grid to software with a synthesis: what computerization imported into the artist's workshop is the logic of the machine — the automatic mode, the preset, the look molded into a single operation, replaced when fashion changes, the way you replace a machine tool gone good for nothing. What it evicted is the logic of the tool: general-purpose controls, and the emphasis on the skill of the hand.

Against the machine, the positive model exists, and music has always known it: the instrument. An instrument is a tool carried to its extreme form: generic (a piano plays everything), transparent in the sense of the previous chapter, and demanding — its difficulty belongs to the music, not to its construction. This is exactly the program I set for my software: to be an instrument of visual expression that lets the artist develop his own

interpretation of the raw material, as a musical instrument lets the musician interpret the score, by giving a wide expressive range; in return, basic theoretical and practical training is expected of the user, and visual quality will not be sacrificed to flatten the learning curve. Research on musical interfaces has stated the design ideal with definitive precision: a low "entry fee" with no ceiling on virtuosity — and it observes that many simple-to-use musical interfaces take on, after brief use, the character of a *toy*[5]. The word is no coincidence: it is the word of the chapter "In praise of the simple". Instrumental interaction is, moreover, an established model in interface research: the instrument as a two-way transducer between the user and the domain object[6] — the formal definition of the remote control from the chapter "The transparent tool". And the philosophy of technology gives the program its most unexpected endorsement: Simondon held automatism to be "a fairly low degree of technical perfection"; the true perfecting of a machine corresponds, on the contrary, to its operation harboring a *margin of indeterminacy*[7]. That margin is precisely where the artist lives. The machine closed over its predetermined operation can only give summary results; the open machine needs the human "as musicians need the conductor" — the musical analogy is in the text — or rather as the instrument needs a musician. There remains practice itself, readily presented as the price to pay before earning the right to create, years of scales as a toll at the gates of music. Sennett shows this is a misreading: at its highest degree, technique ceases to be a mechanical activity, and it is only once you do a thing well that you become able to feel it fully and think it deeply while doing it[8]. Training does not precede expression: it makes it

possible. The successful instrument ends up disappearing into use, like the carpenter's hammer, which exists for him only when it breaks[9]; the tool that second-guesses the artist's intention is, in those terms, permanently broken. There remains the pianist's test, which software still lacks: when you struggle on a concert grand fresh out of servicing, you know the instrument is not to blame; in front of software, it is harder to tell whether the fault lies with the instrument or its operator — unless the tool publishes its contract, its criteria, and its limits, as the chapter "The transparent tool" demands. Good retouchers pull a result out of any software; the difference is the time it costs them, and that time is the measure of the instrument.

The case has a political name, coined by Illich. The convivial tool is one that can be "easily used, by anybody, as often or as seldom as desired, for the accomplishment of a purpose chosen by the user"; between man and the world, it is a "conductor of meaning, a translator of intentionality"[10] — I know no finer definition of the transparent instrument. And Illich named the inverse: "Radical monopoly exists where a major tool rules out natural competence." Transposed to photo software: the major tool that rules out natural competence is the preset and the generated style, which fit the letter of the definition.

The objection left standing is the most sympathetic one: democratization. Automation opens photography to those who have neither the time nor the taste for training; who am I to bar the door? The objection has an instructive precedent. In 1906, John Philip Sousa published "The Menace of Mechanical Music": machines that sing and play the piano "in substitute for human

skill, intelligence, and soul" would make "the tide of amateurism" recede, and the country would stop singing[11]. History mocked him on listening: the record created listeners by the millions, and musicians with them. But on the gesture, his prediction has aged better: the player piano created no pianists, and no one ever called the player piano a democratization of the piano. The distinction settles the objection: listening is consumption, creating is production, and full automation democratizes the former while evicting the latter. The complete answer lies in the musical-interface criterion: lowering the entry fee is legitimate, even desirable — it is the ceiling that is the crime. The preset as a starting point that is inspectable, tunable, bypassable, lowers the entry without capping anything: that is an instrument that helps. The automatic mode without a manual mode caps expression at the average of its training corpus: that is radical monopoly in the strict sense, the major tool ruling out natural competence. The real democratization of an art comes through the entry, never through the ceiling.

Designing for art therefore means building instruments and refusing machines: generic tools rather than molded looks, a margin of indeterminacy rather than predetermined operation, ends chosen by the user rather than executed for him, a low entry and no ceiling. There remains the question this criterion leaves open, and it is the most delicate one in the book: between the molded look that imposes the developer's taste and the forest of sliders that drowns everyone, how does an instrument *open up* possibilities? That is the subject of the next chapter.

1. Russial, J. & Wanta, W. (1998). “Digital Imaging Skills and the Hiring and Training of Photojournalists”. *Journalism & Mass Communication Quarterly*, 75(3) https://doi.org/10.1177/107769909807500312 ; Russial, J. (2000). “How Digital Imaging Changes Work of Photojournalists”. *Newspaper Research Journal*, 21(2), 67-83 https://doi.org/10.1177/073953290002100206 ; Klein-Avraham, I. & Reich, Z. (2016). “Out of the frame: A longitudinal perspective on digitization and professional photojournalism”. *New Media & Society*, 18(3), 429-446 https://doi.org/10.1177/1461444814545289 . These studies establish the fact of the forced conversion and of the shift of the work toward the screen; the judgment passed here is mine. ↩

2. Braverman, H. (1974). *Labor and Monopoly Capital: The Degradation of Work in the Twentieth Century*. New York: Monthly Review Press, ch. 4 — the Taylorist principles, including the separation of conception and execution https://openlibrary.org/works/OL1986011W . Braverman deals with industrial wage labor; the extension to leisure is the author’s. ↩

3. Mumford, L. (1934). *Technics and Civilization*. New York: Harcourt, Brace & Co., pp. 9-12 https://openlibrary.org/works/OL1879251W — “the tool lends itself to manipulation, the machine to automatic action.” ↩

4. Ellul, J. (1954). *La Technique ou l'Enjeu du siècle*. Paris: Armand Colin; repr. Economica, 2008 https://openlibrary.org/works/OL1985863W — ch. II on the characteristics of traditional technique, rendered here in indirect discourse: the text was checked against the English translation (*The Technological Society*, 1964, pp. 64-77; the single-task machine pp. 74-75), not collated with the French edition. The tool/machine division of labor between Mumford and Ellul is deliberate: Ellul explicitly refused the assimilation of technique to the machine, and reproached Mumford precisely for committing it; the synthesis of the two is mine. ↩

5. Wessel, D. & Wright, M. (2002). "Problems and Prospects for Intimate Musical Control of Computers". *Computer Music Journal*, 26(3), 11-22 https://doi.org/10.1162/014892602320582945 — the section is titled exactly "Low 'Entry Fee' with No Ceiling on Virtuosity"; the "toy-like" character of simple-to-use musical interfaces is noted there verbatim. ↩

6. Beaudouin-Lafon, M. (2000). "Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces". *Proceedings of CHI 2000*. New York: ACM, 446-453 https://doi.org/10.1145/332040.332473. ↩

7. Simondon, G. (1958). *Du mode d'existence des objets techniques*. Paris: Aubier, Introduction https://openlibrary.org/works/OL4504172W — "the true perfecting of machines [...] corresponds not to an increase in

automatism but, on the contrary, to the fact that the operation of a machine harbors a certain margin of indeterminacy", automatism being "a fairly low degree of technical perfection"; man is there the permanent organizer of a society of open machines "that need him as musicians need the conductor" (our translation). ↩

8. Sennett, R. (2008). *The Craftsman*. New Haven: Yale University Press, Part I, ch. 1 https://openlibrary.org/works/OL1870880W . On the 10,000-hour measure, Sennett draws on Ericsson's work (Ericsson, K. A. *et al.*, 1993, *Psychological Review*, 100(3), 363-406) https://doi.org/10.1037/0033-295x.100.3.363 . ↩

9. Winograd, T. & Flores, F. (1986). *Understanding Computers and Cognition*. Norwood, NJ: Ablex, ch. 3 https://openlibrary.org/works/OL28171792W — the take-up of Heidegger's hammer: ready-to-hand, the tool disappears from awareness; it becomes present again only at breakdown. ↩

10. Illich, I. (1973). *Tools for Conviviality*. New York: Harper & Row, ch. II for the definition of the convivial tool, ch. III for radical monopoly https://openlibrary.org/works/OL2848898W (French version, *La convivialité*, Paris: Seuil, 1973, established by the author with L. Giard and V. Bardet). "Conductor of meaning, translator of intentionality" renders the French version's « conducteur de sens, traducteur d'intentionnalité » (our translation; the phrase has no counterpart in the English

text). The radical-monopoly sentence is quoted verbatim from ch. III of the English original. ↩

11. Sousa, J. P. (1906). "The Menace of Mechanical Music". *Appleton's Magazine*, 8, 278-284 https://ocw.mit.edu/courses/21m-380-music-and-technology-contemporary-history-and-aesthetics-fall-2009/18ab3aba9fe7aa1502a55cd049333659_MIT21M_380F09_read02_sousa.pdf. His forecast about listening proved wrong; the one about the amateur practice of the musical gesture, much less so. ↩

17. Enabling without holding the brush

The previous chapter left the instrument designer on a narrow ridge between two precipices. On one side, holding the brush in the artist's place: the molded look, the automatic mode, the preset-as-destination — so many ways of embedding in the tool the developer's opinion of what an image should be, and of serving it to thousands of people who never asked for it. On the other, refusing to choose anything at all: dumping the engine's raw parameters, and drowning everyone under the forest of sliders that the chapter "In praise of the simple" put on trial. Most image software oscillates between the two, and much of it stacks them: opaque automations *and* 40 settings. This chapter looks for the crest — how an instrument *opens up* possibilities without imposing a taste or requiring an engineer — and it begins by building the case file: what exactly did the computer do to creation?

What it took away, craftsmanship had named before computing. David Pye distinguished two regimes of making: the workmanship *of risk*, in which the quality of the result is not predetermined but "depends on the judgement, dexterity and care which the maker exercises as he works" — quality continually at risk throughout the act — and the workmanship *of certainty*, that of quantity production, in which the result is settled before the gesture begins, and whose pure form is full automation[1]. The push button is the workmanship of certainty applied to art: the

result preexists the gesture, and the gesture can no longer bring anything to it — nor cost it anything. Now what risk carried away with it was the maker's entire presence. Sherry Turkle documented the mourning among drawing professionals when simulation took over: the MIT architects wanted to preserve "sacred spaces" — hand drawing first — where their intimacy with the object was at stake[2]. And Brian Eno said the same thing as a musician, in a phrase that became famous: "The problem with computers is that there is not enough Africa in them" — and he went on, less quoted: the computer "uses so little of my body. You're just sitting there, and it's quite boring. You've got this stupid little mouse that requires one hand, and your eyes. That's it. What about the rest of you? [...] It's imprisoning"[3]. Three witnesses, three trades, one finding: the computerization of creation was first an amputation — of risk, of the hand, of the body.

It was then a normalization, and that case file opens before the computer, in 1888 precisely. "You press the button, we do the rest": the slogan is George Eastman's own, for the first Kodak — the camera was sent back whole to the factory, which developed, printed, and reloaded[4]. It is the first push-button promise in the history of art, and its balance sheet is double, exactly like that of the chapter "The instrument and the machine": it created popular photography — millions of images, millions of photographers — and it created, in the same gesture, the *vernacular* of the snapshot: the same framings, the same poses, the same subjects, a Kodak culture whose stupefying uniformity ethnography has shown[4]. Vilém Flusser supplied the theory of the phenomenon,

and it became the canonical text of photography: the camera is a program — "every photograph is a realization of one of the possibilities contained within the program of the camera", and the number of possibilities, though large, is finite — and photographers "can only act within the program of the camera, even when they think they are acting in opposition to this program"[5]. The ordinary photographer is the functionary of that program; the experimental photographer is the one who knows he is "playing against the camera", who attempts "to place within the image something that is not in its program" — "Freedom is playing against the camera"[5]. Lev Manovich extended Flusser into the software era: the operations of creative software become the operations of culture itself, and the design question is stated in his terms — how software makes some design choices "seem natural and easy to execute, while hiding other design possibilities"[6]. The platform's filter makes the style of the decade. The evidence converges: when millions of creators receive the same program, the same factory defaults and the same presets, the immense space of possible images folds down onto the small subspace the tool makes easy — a narrowing that has been measured elsewhere, in popular music[7]. The developer's taste, multiplied by distribution, becomes the style of an era — and nobody chose it.

What does the push button do to creativity itself? Psychology supplies two mechanisms, and they converge. Margaret Boden described creation as the exploration, then the transformation, of a conceptual space: *combinatorial* creativity recombines known elements; *exploratory* creativity travels an existing space of

possibilities; *transformational* creativity — the rarest — changes the rules that define the space itself[8]. Read the hierarchy in terms of tools: the tool defines the space the artist can explore. The push button reduces creation to browsing a pre-enumerated catalog — the poorest form of the first level; the open instrument allows exploration; and only the instrument whose program can be overflowed — Flusser again — leaves transformation a chance. The second mechanism is Csikszentmihalyi's: enjoyment "appears at the boundary between boredom and anxiety, when the challenges are just balanced with the person's capacity to act"; too low a challenge produces not comfort but boredom, then disengagement; too high a challenge produces anxiety and the feeling of being overwhelmed; the optimum between the two is named *flow*: a near-second state in which the perception of time is altered, fatigue goes unfelt, the subject is entirely absorbed in the task and draws from it a particular joy.[9] The push button is below the threshold: it asks nothing, so it engages nothing — it is the "virtuosity ceiling" of the musical interfaces of the chapter "The instrument and the machine", rediscovered by psychology. Yet the other half of the picture must be kept, for it is counterintuitive: the entirely blank page does not liberate either. Research on constraints shows it: faced with total freedom, people redo what has already worked; it is the *chosen* constraint that forces you off the reliable answers and pushes toward the new ones — Monet, Stravinsky, and Chanel created *through* their constraints, not despite them[10]. The whole question is who chooses the constraint: the one the artist imposes on himself is an engine; the one the tool vendor imposes is a mold. Eno, who spent his life on both sides, wrote the synthesis: "The trouble

begins with a design philosophy that equates ‘more options’ with ‘greater freedom’” — the relationship between user and machine, he told synthesizer makers, “might be better achieved by reducing options”[10].

From this case file, the ethos follows term by term, and it is the doctrine of my own software. First principle: *axes* of possibility, not looks. An honest instrument does not offer a catalog of results — it offers the dimensions that generate the results. The difference is not one of quantity but of geometry: a catalog is finite, browsable, normalizing; a space with a few axes is continuous, and its points belong to no one. This is where the options dilemma from the opening resolves: the remedy for the forest of sliders is not less control, it is *orthogonality* — perceptually calibrated dimensions that do not fight one another, each doing one thing the eye knows how to name. Forty sliders that interfere are a punishment; four orthogonal axes are an instrument — and building those axes, calibrating their scales on perception rather than on the internal mechanics, is precisely the design work that the push button dodges and that the forest of sliders never did. Second principle: the starting point is *faithful*, never aesthetic. The tool first delivers the scene as measured, shaped by physics and perception — the contract of the chapter “The transparent tool” — and any intent about the image belongs to what follows, which belongs to the artist alone. Third principle: the preset is an example, not a destination — the chapter “The instrument and the machine” laid down its three conditions, inspectable, tunable, bypassable; I add the fourth: *pedagogical* — a preset should open up and be readable, like a score, so that you

learn the path from it rather than camp on it. Fourth principle, and it is the guardrail of all the rest: the instrument states no aesthetic — it makes aesthetics *reachable*. Its success is measured not by the images it favors, but by those it does not forbid: Simondon's margin of indeterminacy, Wessel and Wright's low entry with no ceiling, Flusser's program you can overflow — three names for the same demand. To design for art is to write the richest possible program and step aside for whoever plays it — open the field, without holding the brush. And this demand is not specific to art: making the other capable of what he could not have done alone, then stepping aside, is what we owe anyone who depends on a tool. The artist is merely the case where the stakes show best.

The instrument that achieves this does as much as it can, and then disappears into use; and if it is loved deeply, used habitually, handed down, it has succeeded in its highest form. But an instrument is only handed down if it lasts, and software only lasts if someone finishes it and maintains it. That is the subject of the next chapter.

---

1. Pye, D. (1968). *The Nature and Art of Workmanship*. Cambridge: Cambridge University Press (rev. ed., London: Herbert Press, 1995), ch. 2 https://www.worldcat.org/title/nature-and-art-of-workmanship/oclc/32411126 — workmanship "in which the quality of the result is not predetermined, but depends on the judgement, dexterity and care which the maker exercises as he works [...] the quality of the result is continually at

risk during the process of making" (*workmanship of risk*); the workmanship of certainty, "always to be found in quantity production, and found in its pure state in full automation". ↩

2. Turkle, S. (2009). *Simulation and Its Discontents*. Cambridge, MA: MIT Press, pp. 4-5 https://mitpress.mit.edu/9780262546799/simulation-and-its-discontents/ — the early skeptics "wanted to preserve what they termed 'sacred spaces,' places where technology might disrupt sacrosanct traditions linked to core values. So, for example, architects wanted to preserve hand drawing; they stressed its history, its intimacy, and how it tied architecture to the arts." The field is that of MIT professionals (architecture, engineering, science), not artists in the strict sense; the mechanism described — the loss of intimacy with the object when simulation comes between — is what carries over. ↩

3. Kelly, K. (1995). "Gossip is Philosophy", interview with Brian Eno. *Wired*, 3.05, May 1995 https://www.moredarkthanshark.org/eno_int_wired-may95.html: "The problem with computers is that there is not enough Africa in them"; the version "the trouble with computers", which circulates everywhere, is a corruption. The passage on the body: "What's pissing me off is that it uses so little of my body. You're just sitting there [...] That's it. What about the rest of you? [...] It's imprisoning". ↩

4. The slogan is Eastman's own, for the camera called "The Kodak" of 1888 (the "No. 1" is its 1889 successor): the

camera, sold preloaded with 100 exposures, went back whole to Rochester for developing, printing, and reloading. The ethnography of the vernacular: Chalfen, R. (1987). *Snapshot Versions of Life*. Bowling Green, OH: Bowling Green State University Popular Press https://search.worldcat.org/title/snapshot-versions-of-life/oclc/608210070 — the "Kodak culture": conventions of framing, pose, and subject remarkably uniform across family albums. ↩ ↩

5. Flusser, V. (2000). *Towards a Philosophy of Photography* (trans. A. Mathews). London: Reaktion Books https://archive.org/details/towardsphilosophooooflus, p. 26: "The camera is programmed to produce photographs, and every photograph is a realization of one of the possibilities contained within the program of the camera. The number of such possibilities is large, but it is nevertheless finite"; p. 38: "Photographers can only act within the program of the camera, even when they think they are acting in opposition to this program"; p. 27: "photographers do not play with their plaything but against it"; pp. 80-81: the experimental photographers attempt "to place within the image something that is not in its program. They know they are playing against the camera" — "Freedom is playing against the camera". The French text of this book quotes the French version (*Pour une philosophie de la photographie*, trans. J. Mouchard from the revised 1989 German edition, Belval: Circé, 1996; pocket repr., 2004 https://www.babelio.com/livres/Flusser-Pour-une-philosophie-de-la-photographie/631558), whose wording

differs at points — e.g. « être libre, c'est jouer contre les appareils » (p. 83). ↩ ↩

6. Manovich, L. (2001). *The Language of New Media*. Cambridge, MA: MIT Press https://mitpress.mit.edu/9780262632553/the-language-of-new-media/ — the "Operations" chapter: the operations of software become general cultural operations. Manovich, L. (2013). *Software Takes Command*. New York: Bloomsbury https://doi.org/10.5040/9781472544988 — software as a universal engine of culture; the design question is stated in "Inside Photoshop" (*Computational Culture*, 1, 2011) http://computationalculture.net/inside-photoshop/ . On the aesthetics of platforms: Manovich, *Instagram and Contemporary Image* (online, 2016-2017) https://manovich.net/index.php/projects/instagram-and-contemporary-image . ↩

7. Serrà, J., Corral, Á., Boguñá, M., Haro, M. & Arcos, J. L. (2012). "Measuring the Evolution of Contemporary Western Popular Music". *Scientific Reports*, 2, 521 https://doi.org/10.1038/srep00521 — 464,411 recordings, 1955-2010: measurable homogenization of the timbral palette, restriction of pitch transitions, rising loudness. The study records these trends without identifying their causes; attributing a share of them to production tools is our hypothesis, not theirs. ↩

8. Boden, M. A. (2004). *The Creative Mind: Myths and Mechanisms*, 2nd ed. London: Routledge (1st ed. 1990) https://books.google.com/books/about/The_Creative_Mind.html?id=6Zkm4dz32Y4C — combinatorial, exploratory,

transformational creativity; the conceptual space whose rules define what can be thought. ↩

9. Csikszentmihalyi, M. (1990). *Flow: The Psychology of Optimal Experience*. New York: Harper & Row https://openlibrary.org/books/OL18354762M/Flow, p. 52: enjoyment "appears at the boundary between boredom and anxiety, when the challenges are just balanced with the person's capacity to act". Challenge below skill produces boredom; challenge and skill both low, apathy (ch. 7). ↩

10. Stokes, P. D. (2005). *Creativity from Constraints: The Psychology of Breakthrough*. New York: Springer Publishing Company https://books.google.com/books/about/Creativity_from_Constraints.html?id=XkkmQo10ZVgC — paired constraints: one precludes the reliable solutions, the other points toward the new ones; case studies on Monet, Picasso, Stravinsky, Chanel. Eno, B. (1999). "The Revenge of the Intuitive". *Wired*, 7.01, January 1999 https://www.moredarkthanshark.org/eno_int_wired-jan99.html: "The trouble begins with a design philosophy that equates 'more options' with 'greater freedom'"; as early as the 1995 interview: "I've been telling synthesizer manufacturers for years that the issue is not increasing the number of internal options. The issue is increasing rapport [...] But the relationship between user and machine might be better achieved by reducing options." ↩ ↩

18. Finishing and maintaining

The chapter "Stating the problem" left 3 of the SMART method's 5 letters pending: *assignable*, *realistic*, *time-bound*. They come down to a single rule, which closes this part of the book: **undertake nothing you lack the means to finish and to maintain.** The chapter "In praise of the simple" established that a tool develops until it fulfills its purpose; this chapter says what fulfilling that purpose demands. For finishing is not a state that occurs when you stop working: it is a design object in its own right, to be specified, budgeted and protected — and most software that fails does not fail for having been badly started, but for never having been finished.

For starting was never the problem: starting is euphoria. The fresh repository has no debt, no bugs, no users, no history; the plan looks perfect because nothing has tested it yet, and the code flows — Brooks opened his book on this, the joys of the craft, the pleasure of building castles of pure thought-stuff[1]. This euphoria is a trap with two jaws. The first: it is self-rewarding, when finishing is not — the chapter "What insularity produces" will show a room that applauds announcements and ignores upkeep; you can therefore spend a lifetime starting projects, carried from euphoria to euphoria, without ever shipping anything finished, and feel productive doing it. The second: the hard part is not launching, it is *managing growth* — and that only shows afterward. The working methods that sufficed for one man on

10,000 lines collapse at 100,000 with contributors; the improvised infrastructure becomes the bottleneck; informal communication saturates, chapter "The committee will invent nothing". Growing a project means regularly redesigning the project itself — its tools, its methods, its organization — while it runs, and that kind of design has neither the glory of the beginning nor the comfort of the blank page. The measure of an engineer is not what he starts: it is what he keeps standing over time.

First requirement: you do not program a solution, you develop an architecture. In my software, behind the denoising module — removing the digital grain from a photo taken in the dark — sits a neural network that has to be trained on thousands of images. The network is only the visible part. Around it, I built a complete administration chain: user-side triage of images, so as not to harvest just anything into the training corpus and to filter upstream; maintainer-side validation of contributions; automated retraining of the model that anyone can run by simply clicking a button. The network's architecture itself is nothing novel, and the code embedded in the software is modest; what holds the whole thing together is the work chain around it, the one that spares you checking everything by hand for the next 10 years. You do not ship a fragment of software; you ship the means to work efficiently afterward. That is the criterion that separates the finished from the unfinished: finished work *compounds* — every piece shipped reduces future work. The code fragment does the opposite: it pockets the benefit of the announcement and leaves the burden of operation, and the chapter "Code is a liability" said at what rate that interest runs.

Second requirement: flee whatever demands endless maintenance. The example I regularly hold up against requests: the graphics-card blacklist — cataloging the failing configurations to disable them outright in the code. A blacklist is useless unless it is exhaustive, and to keep it that way you have to maintain it permanently, tracking every combination of card, driver version and operating system: a task that can only grow, without anyone having asked for anything. The moment you hear yourself say "we should have a blacklist," you are about to build yourself a hell: in 6 months it will be forgotten; in two years it will be technical debt that helps no one. The same verdict strikes over-specialized solutions and chains of hand-stitched exceptions. What actually guarantees robustness is tests — and you only test well a reduced set of cases, which loops back to the praise of the simple: the niche feature with the bizarre heuristic is almost never covered by tests, and the day it breaks, nobody sees it coming. Maintainability is decided at the moment of accepting or refusing the feature, not at the moment of repairing it.

Third requirement: launch nothing you cannot carry to completion in the allotted time, and the measure of this requirement is the foundation worksite. Replacing a brick everything else depends on — the graphics library that draws every button and every window — is not an update: between two major versions of such a library, the incompatibilities are not function renames but paradigm shifts. Carrying such a port through demands freezing every feature addition until it is done, for continuing to build modules and interfaces while the base is being replaced amounts to having one worker dig a hole while

his colleague fills it back in. Darktable undertook to port its interface from GTK 3 to GTK 4 without that freeze; the worksite had been dragging on for more than two years as I write this, and that is not bad luck, it is arithmetic: every feature added during the port is extra work to port, produced at the rate the port advances[2]. A project without governance cannot decree a freeze — no one there has the mandate to say no — and that is one of the reasons why the question of organization, which occupies the rest of this book, is not separable from the technical one.

These 3 requirements share a common denominator, which makes them non-negotiable: the maintenance budget, which is to say humans. The surveys of free-software maintainers give the state of the fleet: about 60% are not paid for this work, a comparable proportion has left their project or considered doing so, and nearly one in two cites exhaustion among their reasons[3]. That is the budget drawn on by the blacklist "that would just need keeping up to date," the fragment shipped "to be finished later," the port nobody froze. Every maintenance commitment made lightly is a withdrawal from an account already overdrawn, and the overdraft is paid in burnout — the subject will return, in its own right, in the chapter "The invisible worker". And the balance of this book depends on it: to hold that we owe something to the user obliges us to the same toward the one who keeps the tool standing, failing which we will only have moved the exploitation one notch over.

The objection across the aisle is a shop-floor slogan: *ship it now, fix it later*; the perfect is the enemy of the good; real artists ship. What is true in the slogan is what the chapter "Decide, then hold"

established: you must ship early to measure, and iteration beats infinite specification. But the slogan trades on a slide: shipping a *minimal working solution* early is not shipping an *unfinished fragment*. The minimum viable includes the chain that makes the thing operable and maintainable — the denoiser without its administration chain is not a minimal product, it is a tech demo with a handle screwed on. And "later" is a loan: it assumes a future repair budget, drawn on the account we just saw — unpaid, burned out, resigning. In industry, *fix it later* is sometimes a fundable bet; in community software, it is almost always a euphemism for *never*. The honest version of the slogan exists, and this book defends it: ship early what is finished, small but whole — and refuse to confuse modesty of scope with incompleteness of content. The nuance is subtle, and visible only with experience.

Finishing and maintaining close the loop opened in the chapter "Stating the problem": the problem stated, measured, decided, built, must be *held over time*, and time is budgeted like everything else. But holding over time, for software, takes a precise social form: the stable release, shipped to a public that will live in it. Shipping is not dumping, and that is the subject of the next chapter, which opens the part of the book devoted to the collective.

---

1. Brooks, F. P. (1975). *The Mythical Man-Month*. Reading, MA: Addison-Wesley https://openlibrary.org/books/OL5044740M/The_mythical_man-month, ch. 1, "The Tar Pit," section "The Joys of the Craft"

— the joys of the programmer's craft, followed immediately by its woes: the entire chapter is built on that contrast. ↩

2. Darktable-org/Darktable repository, issue no. 15920 "Gtk4 Migration - Progress" https://github.com/Darktable-org/Darktable/issues/15920, opened December 21, 2023 and still open at the time of writing — more than two years. The porting work proper only began landing in the fall of 2025 (some ten "gtk4-prep" pull requests still open in mid-2026): the worksite thus remained declared, but dormant, for most of those two years. Over the same interval, the project shipped 6 feature releases on its six-month cadence (4.6 in December 2023, up to 5.6 in June 2026) — direct proof of the absence of a freeze. An independent assessment filed in the repository in March 2026 (issue no. 20433 https://github.com/Darktable-org/Darktable/issues/20433) puts the codebase at about 506,000 lines of C, of which some 205 files touch GTK directly, and describes the migration as "a large, high-risk, multi-month undertaking". ↩

3. Tidelift, *The State of the Open Source Maintainer*, 2023 and 2024 surveys https://www.tidelift.com/open-source-maintainer-survey-2024: about 60% of surveyed maintainers are not paid for this work; 58 to 60% have left a project or considered doing so; 44% cite burnout (2024). The share of unpaid maintainers was 46% in 2021 — the trend is worsening. ↩

Part IV — The collective

19. Ship, don't dump

A stable release is a finished feature set. It is not the state of the software frozen at an arbitrary date. The difference between the two definitions looks administrative; in reality it decides who pays the price of the release, and this chapter, which opens the part of the book devoted to the collective, is entirely contained in that question. The trajectory we are about to quantify is in no way unique: it awaits every volunteer project that ships on a fixed date, and it is quantified precisely so that it can be recognized in time. Here is the model I am fighting: open every workstream at once without ever guaranteeing a coherent state, then decree "stable" the main branch frozen in whatever state it happens to be on the agreed date. The release is not driven by a finished feature set but by a calendar; if an important feature is not ready, too bad, it will come later. The symptom of this model has a name in my experience: the post-release rush. A torrent of reports on the bug tracker right after the release, a botched point release two weeks later. Traditionally, the rush happens *before* the release: you wrap up, you test, you ship, and you can finally breathe. Inventing the after-rush means extending the exhaustion past the finish line — you cannot even take a vacation telling yourself it has shipped. And I draw from it the reading that

seems obvious to me: a tracker that explodes at the moment of a "stable" release means the release was not stable.

The opposite discipline existed, and I saw it work before I saw it abandoned. It is the release candidates: the feature set is frozen, you announce publicly that nothing new will land in this release, and the successive candidates — RC0, RC1, RC2 — are nothing more than bugfix milestones, until the last candidate becomes the stable release because there is nothing left to fix. The freeze does all the work: it turns the release into a state you reach rather than a deadline you endure. Darktable practiced that discipline; it was abandoned when release management changed hands, in the name of continuous integration and agile methods, a common reading of which holds any freeze to be slowness. The cost of giving it up is human before it is technical: something breaks 15 times a day, and the reports fly off in every direction. To which add a lesson about testing, counterintuitive and measured. The traditional beta tester is technically advanced: he knows how to write a usable report. Opening testing to the general public too early harvests *more* reports of *lower* quality — not through people's fault, but because they lack the means to describe what they have in front of them. The research on bug reports confirms the mismatch: what developers value most — reproduction steps, execution traces, test cases — is precisely what ordinary reporters find hardest to provide[1]. The effort gets diluted: instead of fixing bugs in tandem with those who know how to describe them effectively, you exhaust yourself decoding the reports of those who do not. The practical conclusion: stay in the laboratory longer, with a chosen panel of testers, and leave the daily builds

to those who choose to live dangerously, on the development branch.

On this subject, the scientific literature seems to conclude otherwise — but we will see that it studies a different object. Martin Michlmayr, in the field's reference dissertation, studied release management in the big free software projects — Debian, GNOME, the Linux kernel, OpenOffice.org, X.org — and concludes that *time-based* releases work there as a coordination mechanism: in a project fed by contributor flows that nobody controls, the calendar is the only conductor available, and quality improves as a result[2]. Thus, when Firefox switched in 2011 to rapid scheduled releases, the study closest to a controlled test of my "torrent of bugs" thesis found no significant increase in post-release bugs — bugs showed up earlier, fixes arrived faster[3]. The predicted torrent did not materialize in the counts — at Firefox. It materialized on my ground, and the two findings do not cancel each other out: they indicate that the results of this literature do not seem to apply to projects structured like Darktable, for causes that remain to be explored — the rest of this chapter proposes some. And the lesson cuts both ways, for this is where the cargo cult prowls: imitating the outward forms of a success in the hope of reaping its effects. Adopting Mozilla's calendar without Mozilla's means is copying the form without the function; what works best for some is not the optimum for all.

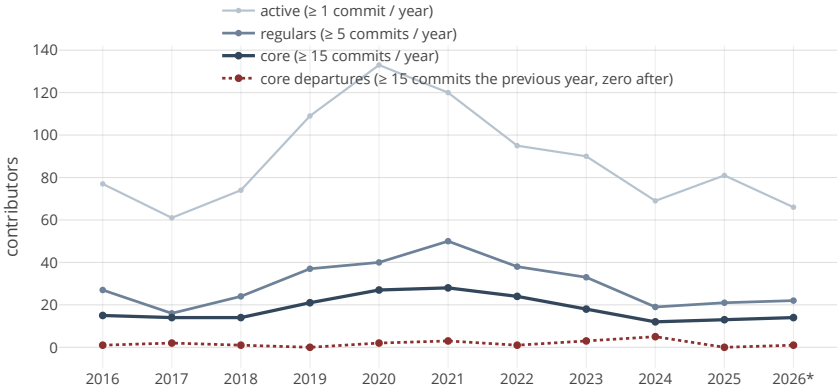
First, two variables these studies do not measure, and which ground the position. The first is the human cost: Michlmayr measures coordination, Khomh counts defects; neither measures the sustainability of the work — the post-release rush, the

impossibility of closing, the exhaustion prolonged past delivery. It is a variable absent from their protocols, not a variable refuted; and the previous chapter gave the state of the human budget it weighs on. The second is architectural capacity: a release cadence without a freeze window structurally forbids foundational worksites — nothing big can ever land, since nothing ever stops. The GTK 4 port, open for more than two years, chapter "Finishing and maintaining", is not an anecdote against time-based releases: it is their mechanical consequence in a project that does not have Mozilla's means. Next, the literature itself has documented the cracks in its own narrative. Over more than 300,000 test executions at Firefox, Mäntylä and his colleagues showed that rapid releases *shrink* the testing scope and *erode* the community of volunteer testers — Mozilla had to hire subcontractors to compensate — while fixing proportionally fewer of the reported bugs[4]; and da Costa measured that, under rapid releases, fixes take a median of some fifty days longer to reach users[5]. The rapid-release narrative thus holds on the defect counts, and cracks on volunteer testing, fix rate and latency — which is to say, on exactly the resources a community project lives on.

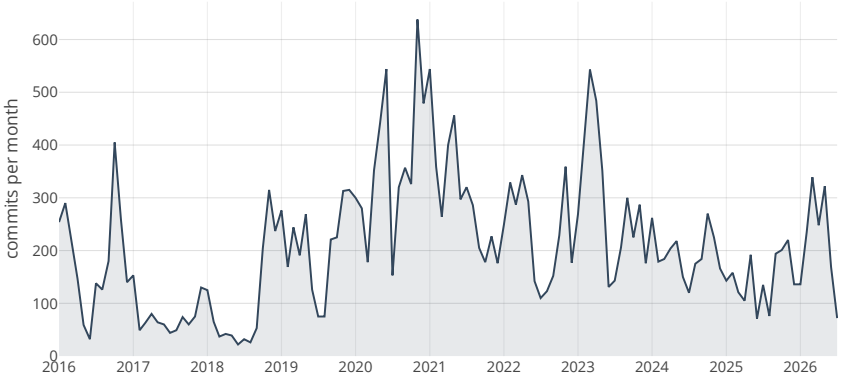
That literature never faced the contradiction with the evidence in hand. Darktable switched to 2 fixed-date releases a year at the end of 2019, and its public records make it possible to follow, over 10 years, what the cadence did to the 3 populations a project lives on: contributors, testers, and users[6].

The hands must be counted first. The number of active commit authors per year fell from a peak of 133 in 2020 to less than half that today — and if you keep only those who contribute at least 5

commits in the year, the regulars went from 50 in 2021 to about twenty since 2024; at 15 commits a year and up, the core that actually carries the code fits in a dozen people. The attrition that produces this melt can be measured too: every year, members of the core — 15 commits or more the previous year — disappear: 9 over the 2022-2024 period alone, including 5 in 2024, more than a quarter of the headcount at that level; and over 10 years, 14 of the 18 leavers never came back[6].



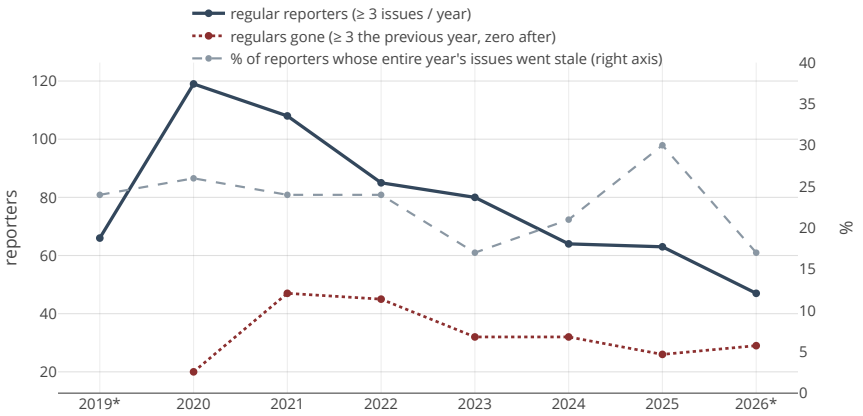
Headcount and departures, 2016-2026: active contributors per year and by intensity (≥ 1, ≥ 5 and ≥ 15 commits in the year), and departures from the core — 15 commits or more one year, zero the next (red dotted line). Commits excluding merges, deduplicated authors; 2026 is a partial year, cut off at July 22. Every curve peaks in 2020-2021 then melts by half; the 2022-2024 spike in departures (9, including 5 in 2024) precedes the low-water mark.



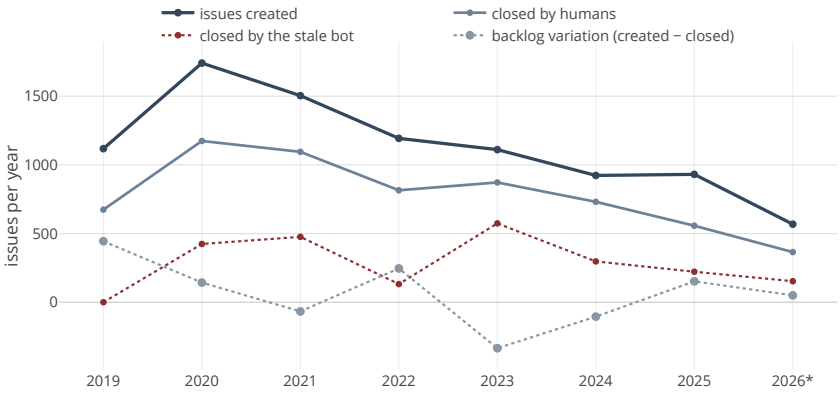
The 10-year pulse: commits per month, 2016-2026 (excluding merges). The crests are release preparations; the absolute summit is late 2020 (638 commits in November). Since then, the amplitude has been melting: the average of the last 12 months (196 commits) is barely more than half that of 2020-2021 (337).

The testers, for their part, can be tracked one by one, and the result matches the literature above with embarrassing precision. Regular reporters — 3 issues or more in the year — numbered 119 in 2020; they are 63 in 2025, and the turnstile spins at full speed: every year, 30 to 45 of them disappear, on the order of 40% of the headcount, replaced by newcomers who will not last any longer. And here is the measurement that explains why they do not last: one reporter in 4, year in year out, sees the *entirety* of the problems he reported during the year end up closed automatically by the bot that sweeps inactive bug reports — for a quarter of the testers, the project's answer to the whole of their year's contribution will have been silence, then the automaton;

the 2025 cohort is already at 30%. Nobody tests for long for a project that does not answer, and the quality of the reports dies with the relationship: the share of issues that never receive a single comment has doubled since 2019[7].



The testers who leave: regular reporters (≥ 3 issues in the year), regulars gone the following year, and share of reporters whose entire year's issues ended up staled by the bot (right axis). The GitHub tracker is born in 2019 (the 2019 sample covers 90% of the year, an API ceiling); 2026 is partial. Tester peak in 2020, half lost since, a turnover of about 40% a year.



The decline of closure capacity, 2019-2026: issues created, closed by humans, closed by the stale bot. Human issue closures drop 53% between 2020 and 2025 — at the exact pace of the working core's melt (−52%) — while the incoming flow recedes by only 47%: the gap is filled by the bot and by the backlog. The dotted line gives the backlog's annual variation (created minus closed, bot included): positive, the arrears grow; negative, they get purged — and they only get purged en masse by the bot (2023). Since 2025, the backlog has been swelling again; cumulative since 2019: about 530 additional issues of arrears.

The series, moreover, bears a dated scar, and I can sign it: 2022. Human issue closures plunge there by a quarter in a single year — from 1,095 to 815, the sharpest drop of the entire series — while the headcount, that year, had so far only slightly declined. It is the year I have testified was the first in which I could hardly land a fix anymore, even though I had 4 years of full-time experience on the project: every bug I read pointed to highly

random behaviors, handled by source files that had become incomprehensible. My annual commits go from roughly 200–280 between 2019 and 2021 to fewer than 80 in 2022; those that mention a fix, from about 50 a year to 19[8].

The post-release rush itself can be measured. Around each major release, I counted the fix commits in the 30 days before and the 30 days after the release. The verdict has two eras:

- **Up to mid-2021**, the after exceeds the before by 27%: the torrent I was describing did exist. The 3.4 of December 2020 is its summit — 98 fixes the month before, 151 the month after.
- **Since the 3.8 of December 2021**, the ratio inverts: post-release fixes run a third below pre-release ones. But…

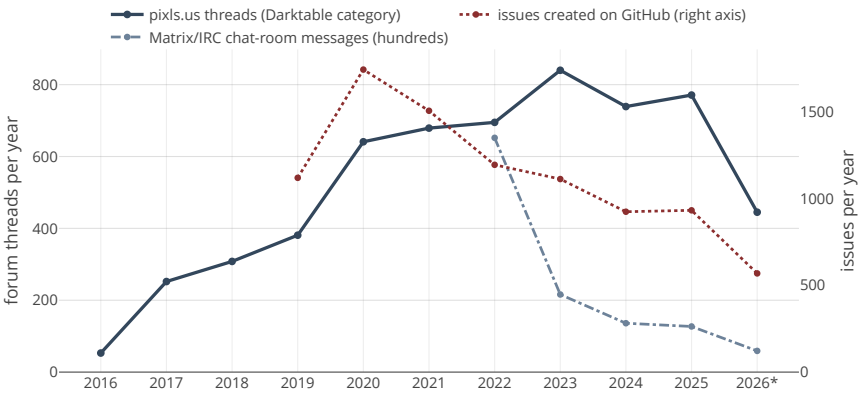
The inversion is not a cleanup — the issue backlog and the latencies say otherwise. The capacity to rush at the bugs simply vanished with the headcount: a rush takes hands, and they are gone. The post-release rush has not been solved; it has been pushed further down the queue[9].

The simultaneous decline in issue openings and closures could be explained otherwise. First hypothesis: the issues of old were inflated with feature requests, since handled, leading to software that is now complete — the labels refute it: requests labeled as such never exceeded a few dozen a year, stable, out of a thousand entries. Second hypothesis: users report less because everything works better — the reporter survey says the opposite, and it says who left: on a constant sample of 300 issues per year, the number

of distinct reporters *rises* (129 in 2019, 215 in 2025) while the number of issues per reporter *falls* (2.3 to 1.4) — the assiduous reporters, those who followed up on their reports and filed many, have disappeared; what remains is the passerby's isolated report, three-quarters of the time from a first-time reporter, every year. It is the most invested who have gone silent — the withdrawal of voice, measured at the counter. Third hypothesis: closure capacity is falling faster than the flow — correct: human issue closures (bot excluded) melted by 53% between 2020 and 2025, at the precise pace of the working core (−52%), while the incoming flow fell by only 47%; the gap is filled by the bot, whose share holds steady, and by the backlog, which is swelling again. And the door must be closed on a reassuring reading: these two declines are not parallel. Relative to staff, closures per head are constant — 43 per person per year in 2020, 43 in 2025: those who stayed absorbed nothing of the loss, capacity tracks headcount, exactly. Relative to the flow, they recede: humans closed the equivalent of 67% of the incoming flow in 2020, 60% in 2025 — if the drop in reports were a relief, that rate would have risen; it fell. A team cut in half, at unchanged individual output, no longer keeps up with even a flow cut in half. Fewer issues because those who believed in it left; fewer closures because those who closed left; and the difference rots or gets swept away by the automaton[10].

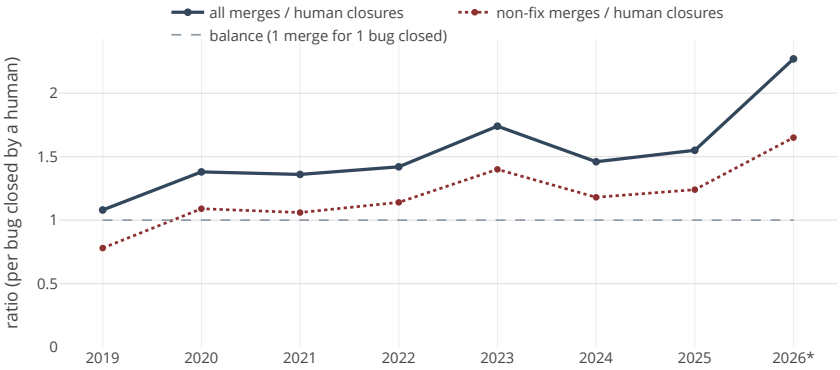
An external cross-check completes this test. If users were reporting less because they are fewer than before, or less engaged, their forum would show it. The opposite is true: on discuss.pixls.us, the Darktable category has never been more alive — 641 threads opened in 2020, 840 at the 2023 peak, 771 in

2025, and 2026 keeps the same pace; messages there are hitting records. While the issues melted by half, the users' conversation did not flag by a single thread, and new features kept pouring into the software, which seems incompatible with a stabilization. The decline is therefore not the community's: it is that of the channel linking it to development. Users have not left the software — they have stopped talking to its developers[11]. And a third venue confirms which side the decline is on: the project's own chat room — the Matrix/IRC channel where developers and regulars talk to each other — went from 65,000 messages in 2022 to a pace of about 10,000 a year in 2026, an 85% melt; to the point that one of the chat room's main "interlocutors" today is the automaton that announces commits[12]. The users' forum lives; the tracker and the developers' room are emptying out: it is not the public that is missing, it is the house that no longer receives.



The decoupling, across 3 venues: discussion threads opened in the Darktable category of discuss.pixls.us (left axis) against issues created on the GitHub tracker (right axis) and messages in the project's Matrix/IRC chat room (in hundreds, left axis). The user community is not declining — the channel linking it to development, and the developers' own conversation, are. The GitHub tracker is born in 2019, the chat-room log starts in February 2022 (its first year is thus already truncated by 6 weeks); 2026 is partial.

Finally, the flow of new features has not slowed at the same pace as the repair of bugs. The composition of the merged flow of pull requests completes the picture: the share of code additions in it has stayed stable, around 20%, while human bug closures collapsed — the ratio of merged novelty to bugs closed by humans thus went from roughly one to one in 2020-2021 to 1.5 to one in 2026. The project merges novelty faster than it repairs the existing, and the gap is widening — counting all merges, fixes included, the ratio reaches 2.3 to one over 2026[7].



Merging faster than repairing: number of merged pull requests for every bug closed by a human (bot excluded), counting all merges and then only non-fix merges ('fix' absent from the title). Below the balance line, the project repairs more than it adds; above it, it goes into debt. 2026 is cut off at July 22.

It remains to say what this rhythm does to people — for the records measure volumes, not fatigue. A merged feature has two recipients. For the developer, it becomes next year's maintenance burden — the chapter "Non-heroic ambition" prices what a feature costs after its delivery. For the invested user, the one who knows the tool and uses it to work, it undoes part of what he had learned. The two recipients thus pull on the same rope, through opposite mechanisms: one maintains what he did not ask for, the other unlearns what he knew. If the hypothesis holds, it explains what the headcount decline alone does not — why the most invested are the first to leave, and on both sides at once. What the user loses in the operation has, moreover, an accounting name: years of learning, a library of presets, a workflow — capital he

built up by trusting. Destroying it inadvertently is the first of the failings, and the hardest to repair.

The literature had written this script before the data. Mäntylä, cited above, had measured it: fixed-cadence releases shrink the testing scope and *erode the community of volunteer testers* — Darktable switched to fixed-date releases at the end of 2019, the tester peak is in 2020, half have disappeared since: it is the trajectory measured at Firefox, without the subcontractors Mozilla hired to compensate. Bettenburg, cited above, had shown that the bug report lives on the dialogue between reporter and developer; the silence served to a quarter of the cohorts smothers that dialogue at the root. Hirschman, in the chapter "The closed circle", explains the order of the departures — the most invested, the most demanding about quality, exit first: they are exactly the assiduous reporters of the survey above. And we hold, to finish, the measured version of a dogma that the chapter "Thinking against oneself" will dismantle: Linus's law promised that given enough eyeballs, all bugs are shallow. The record keeps count of the eyeballs: it is melting.

The resolution of the problem is a division of regimes, and it bounds my own thesis along with theirs. The list of projects studied by the literature is not homogeneous, and covers projects that are free software, certainly, but backed by companies and foundations employing salaried staff. Only one of its members is a production application comparable to editing software (the OpenOffice office suite). Debian does not design software in this book's sense: a distribution assembles parts produced elsewhere and solves dependency sets — integration work, not design. The

Linux kernel and X.org, the display server, are infrastructures with no user interface of their own: they are tested machine against machine, with no panel and no habitus. GNOME does design a desktop environment, but backed by the salaried staff of the big commercial Linux distributors. Mozilla Firefox is content-consumption software (a web browser), not creation software. Michlmayr's coordination argument therefore holds for what he studied: massive, uncontrolled contributor flows, on objects that integrate or test mechanically, with salaried staff at the center. But a small volunteer team that *designs* an application with an interface, controls its scope and can freeze what it has promised, is within the field of none of these studies — neither by the nature of the object, nor by the scale of the headcount, nor by the status of the workers; for such a team, releasing by feature set, held by the discipline of the candidates, remains defensible — and that is where I work. The symmetrical error would be to import the cathedral's calendar into the workshop, or the workshop's discipline into the cathedral. My position is not a rival thesis: it is the case these studies did not look at — and it is the ordinary case, for a free software project is far more likely to go through a Darktable phase than a Mozilla phase; the salaried giants are the ecosystem's exception, not its rule. The Darktable case shows that conclusions drawn from Debian or Firefox do not generalize to all free software, and the problem lies upstream, in the very idea of comparing projects that have almost nothing in common except the license. "Free software" is a legal category in intellectual and industrial property law, not a category of organization; as long as people sample within it as within a

homogeneous object, the results will hold for the projects that resemble the samples, and for them alone.

Ship, don't dump: deliver a finished whole to people who are going to live in it, instead of dumping overboard whatever the calendar found on the branch. The question that follows is that of the sentence's subject: *who* decides what is finished, who freezes, who holds the freeze? One would like to answer: the collective. We must look at what collectives actually know how to decide, and that is the object of the next chapter: the committee will invent nothing.

---

1. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R. & Zimmermann, T. (2008). "What Makes a Good Bug Report?". *Proceedings of FSE 2008*, ACM https://doi.org/10.1145/1453101.1453146 — survey of 466 Apache, Eclipse and Mozilla developers: reproduction steps, traces and test cases are the most valued items and the hardest to obtain from reporters ("information mismatch"). ↩

2. Michlmayr, M. (2007). *Quality Improvement in Volunteer Free and Open Source Software Projects: Exploring the Impact of Release Management*. Doctoral thesis, University of Cambridge — case studies: Debian, GNOME, Linux, OpenOffice.org, Plone, X.org https://www.cyrius.com/publications/michlmayr-phd/ . See also Michlmayr, M., Fitzgerald, B. & Stol, K.-J. (2015). "Why and How Should Open Source Projects Adopt Time-Based

Releases?". *IEEE Software*, 32(2), 55-63 https://doi.org/10.1109/MS.2015.55 . ↩

3. Khomh, F., Dhaliwal, T., Zou, Y. & Adams, B. (2012). "Do faster releases improve software quality? An empirical case study of Mozilla Firefox". *Proceedings of MSR 2012*, IEEE https://doi.org/10.1109/MSR.2012.6224279 ; extended version: *Empirical Software Engineering*, 20(2), 2015 https://doi.org/10.1007/s10664-014-9308-x . No significant increase in post-release defects when Firefox switched to rapid releases; defects show up earlier, fixes are faster. ↩

4. Mäntylä, M. V., Adams, B., Khomh, F., Engström, E. & Petersen, K. (2015). "On rapid releases and software testing: a case study and a semi-systematic literature review". *Empirical Software Engineering*, 20(5), 1384-1425 https://doi.org/10.1007/s10664-014-9338-4 — 312,502 test executions, Firefox 2006-2012: shrunken testing scope, eroded volunteer-tester community (recourse to subcontractors), smaller proportion of reported bugs fixed. ↩

5. da Costa, D. A., McIntosh, S., Kulesza, U. & Hassan, A. E. (2016). "The Impact of Switching to a Rapid Release Cycle on the Integration Delay of Addressed Issues". *Proceedings of MSR 2016*, ACM https://doi.org/10.1145/2901739.2901764 — under rapid releases, integrated fixes take a median of about 50 days longer to reach users. ↩

6. Data cut off at July 22, 2026; https://github.com/Darktable-org/Darktable repository (GitHub API and full git history,

commits excluding merges, authors deduplicated by name; the detail of the daily flow and of automated staleness is in the chapter "The committee will invent nothing"). Headcounts 2016-2026: ≥ 1 commit/year: 77, 61, 74, 109, 133, 120, 95, 90, 69, 81, 66*; ≥ 5: 27, 16, 24, 37, 40, 50, 38, 33, 19, 21, 22*; ≥ 15: 15, 14, 14, 21, 27, 28, 24, 18, 12, 13, 14* (2026 partial). Core departures (≥ 15 commits in year n−1, zero in year n): 1, 2, 1, 0, 2, 3, 1, 3, 5, 0, 1*; 4 of the 18 leavers of 2016-2025 eventually committed again. Monthly pulse: n = 26,865 commits, 2016-2026. ↩ ↩

7. Full sweep of the issues created from 2019 to July 22, 2026 (n = 8,971; the API caps the 2019 sample at 1,000 out of 1,118, i.e. 90% of the year). The bounds are not chosen: Darktable migrated its bug tracking from a self-hosted Redmine to GitHub at the end of 2018 — 13 GitHub issues in 2018, 1,118 in 2019 — and the old Redmine, still browsable in 2019, has redirected to the project's site since December 2019 (Wayback archives of redmine.Darktable.org: last capture served in November 2019 https://web.archive.org/web/20191117225329/https://redmine.Darktable.org/). There is therefore no GitHub archive prior to 2019, and the Redmine records were not carried over. Regular reporters (≥ 3 issues/year): 66*, 119, 108, 85, 80, 64, 63, 47*; gone (≥ 3 in year n−1, zero in year n): 20, 47, 45, 32, 32, 26, 29 (2020 ⟶ 2026); share of reporters whose entire year's issues carry the stale label: 24, 26, 24, 24, 17, 21, 30, 17% (the young cohorts have not finished aging); issues with no comment at all: 3.4% (2019) ⟶ 6.0% (2025) and 12.5% over 2026 so far; on the other hand, on a sample of 68 issues

spread over 2019-2025, 93% received at least one first human reply — it is the follow-up that dies, not the first contact. Merged pull requests carrying "fix" in their title (title heuristic): 28, 21, 22, 20, 20, 19, 20, 28% (2019 → 2026); ratio (non-fix merges) / (human closures): 0.8 (2019); 1.1; 1.1; 1.1; 1.4; 1.2; 1.2; 1.65 (2026); ratio (all merges) / (human closures): 1.1; 1.4; 1.4; 1.4; 1.7; 1.5; 1.6; 2.3. ↩ ↩

8. Human issue closures 2021, 2022, 2023: 1,095, 815, 872; bot closures: 476, 132, 574; core at 15 commits and up: 28, 24, 18. My commits in the Darktable repository (excluding merges): 197 in 2019, 281 in 2020, 210 in 2021, then 66 to 78 in 2022 depending on the alias-matching method; those whose message mentions a fix ("fix"): 76, 57, 46, then 19. The chapter "The closed circle" dates and documents the conflict. ↩

9. 30-day windows around each X.Y.0 release (repository tags), commits excluding merges whose message mentions a fix, totals deduplicated by release date: 433 before versus 548 after from 2.2 (2016) to 3.6 (June 2021), i.e. +27%; 413 before versus 273 after from 3.8 (December 2021) to 5.6 (June 2026), i.e. −34%. First point release 8-84 days after the release (median ≈ 55). ↩

10. Tests of July 22, 2026. Feature requests: "feature: new" label on the issues created per year, 2019-2025: 9, 11, 38, 18, 35, 29, 22 — never more than 4% of the flow (limitation: 44% of recent issues are unlabeled; a GitHub discussion space also exists). Reporters: constant sample of the first 300 issues of each year, 2019-2025; distinct reporters: 129, 165, 160, 183,

172, 203, 215; issues per reporter: 2.33, 1.82, 1.88, 1.64, 1.74, 1.48, 1.40; share of first-time reporters stable between 72 and 83%. Capacity: human closures (total minus bot) 1,174 in 2020 → 557 in 2025 (−53%); core at 15 commits and up: 27 → 13 (−52%); issues created: 1,741 → 931 (−47%). Derived: human closures per core member, 43.5 (2020) and 42.8 (2025) — constant to within 2%; coverage (human closures relative to the issues created the same year): 67% (2020) → 60% (2025); the closures of 2025 represent 32% of the incoming flow of 2020. The drop in coverage is not a sampling artifact, and the test can be redone in your head. Question asked: if the propensity to close had not changed and only the flow had fallen, what would be the probability of observing a coverage gap this large by mere chance? In 2020, 1,174 human closures for 1,741 creations: 67.4%; in 2025, 557 for 931: 59.8% — a gap of 7.6 points. Under the hypothesis of a common propensity (64.8%, pooling the two years), the fluctuation that chance alone produces on samples of this size — the standard error of the gap — is 1.9 points. The observed gap is thus z = 3.9 standard errors, and the probability of a gap at least this large by chance, in either direction (two-tailed test on two proportions), is $p \approx 9 \times 10^{-5}$ — less than one chance in ten thousand. The drop in closures is therefore not explained by the drop in reports alone. Acknowledged approximation: coverage relates a year's closures to the same year's creations, whereas a closure may target an older issue; the test treats this ratio as a proportion. ↩

11. Full count of the threads in the Darktable category of the https://discuss.pixls.us/c/software/Darktable/19 (Discourse API, 5,804 threads as of July 23, 2026), by year of creation: 53, 252, 308, 381, 641, 679, 695, 840, 739, 771 then 445 for 2016 ⟶ 2026 (partial — an annual pace of about 800). Caveats: the category mixes peer help, showcases and general discussion; messages, counted by the thread's creation year, accumulate over time. The comparison bears on the trends: forum stable or growing, tracker down by half. ↩

12. Log of the project's public Matrix room (bridge of the IRC channel), export of July 23, 2026, 118,996 messages — detailed provenance in the chapter "The closed circle". Messages per year: 65,151 (2022, from February 16), 21,639 (2023), 13,629 (2024), 12,669 (2025), 5,908 as of July 23, 2026. The year 2022 is inflated by the conflict that occupies the part "Freedom and its conditions" of this book — but 2023, an already "quiet" year, remains triple 2026, and the slope never reverses. Among the top 8 posters of 2025-2026 is the commit-announcement bot. ↩

20. The committee will invent nothing

The previous chapter ended on a question: who decides what is finished, who freezes, who holds the freeze? Free software has a ready-made answer, and it has the beauty of ready-made answers: everyone. The community will discuss, the committee will settle it, democracy will do the rest. I spent years inside that answer, and this chapter is the record of what it produces — not to put the past on trial, but to give any project that believes itself structureless the means to see the structure it already has.

You might think that by seating ten, twenty, or thirty subjectivities in the same room, you would get, by averaging, some kind of global objectivity. You get nothing of the sort: you get the sum of ten, twenty, or thirty personal preferences — most often, the solution each person has already practiced or seen elsewhere, served up again out of laziness or to feel relevant again. Design by committee always ends the same way: people vote for the drawing that collected the most "likes," because no one knows how to get past individual subjectivities and build a viable general solution. Yet the chapter "Stating the problem" described what that overcoming requires: a stated problem, criteria written before the solutions, a session prepared and facilitated. The committee replaces none of these things; it replaces them all with a ballot.

First, measure what the ballot cannot carry, even an honest one, even an informed one. Suppose one vote prefers red text, and another vote, a month later, prefers a red background. Will you write red text on a red background? Each of the two decisions won its ballot; their sum is unreadable. The flaw is not in the voters, it is in the object: you can only vote on units — a button, a color, a module — while the quality of a design lives in the overall coherence of the whole, which is not a unit but a relation among all of them. Coherence cannot be put to a vote, and this is not a practical difficulty, it is a logical impossibility: what would the question be? "Do you vote for everything going together?" Everyone votes yes, and the designer knows no more about what to do next. Coherence is obtained otherwise: through a plan, held over time, and through a protocol that forces you to think against yourself — the chapter "Thinking against oneself" takes its inventory. But a committee, especially one without direction or structure, has neither plan nor protocol: it has agendas.

Next, measure what the ballot smuggles in. In the room, voices never weigh their nominal weight: invisible stripes — seniority, reputation, self-assurance, flight hours real or supposed — mean that people rarely vote on the drawing and often on the drawer, and the rest of this chapter will give Freeman's verdict: the absence of structure does not remove that power, it makes it invisible. The measuring instrument fits in one question: would each member decide the same thing alone as in session? Social psychology has been answering no since the 1950s. In Asch's conformity experiments, about a third of the answers followed the group's unanimous opinion against perceptual evidence

sitting right before the subjects' eyes, three-quarters of the participants yielding at least once; and a single declared dissenter was enough for conformity to collapse[1]. A committee that converges is therefore not necessarily a committee that thinks: it is often a committee where no one wanted to be the dissenter — Janis's groupthink, which we will meet again below, is just this mechanism become culture.

Hence a conclusion to state exactly: **democracy on design details is not undesirable, it is impossible**. The object of the vote does not exist: the detail voted on is a unit, quality is a relation. What can be voted on — the chapter "The word and the thing" established this — are the criteria and the priorities: the goal, not the stroke. And the solution has nothing seductive about it, which is precisely how you recognize it — it is not called freedom, or initiative, or creativity, or collective intelligence; it is called structure, method, and protocol. Name the goal, name the means, write the criteria before the solutions, then stay lucid about the protocol itself, which is a fallible part like any other: a protocol whose limits you do not know is not a method, it is one more superstition.

Against the ballot, people then set the opposite hope: stop deciding anything, and let numbers do the work — add people until collective intelligence emerges. We must look at where this hope draws its authority. "Studies show that the group is smarter than the individual": that, in general, is the optimistic, likable belief free software tries to instill in its collectives — not a title, not a condition, not a task. It is the cargo cult of the chapter "The cargo cult" applied to science: keep the incantation, throw away

the protocol. People remember that studies exist; they have forgotten what those studies measured, on whom, under what conditions, and for how long — and they conflate years of software design with the one-hour exercises the studies rest on. The real record exists, and it says the opposite of what it is made to say.

Collective intelligence exists: across 699 people split into groups of two to 5, the founding study on collective intelligence brought to light a general factor of group performance, just as one exists for individuals. But this factor is only weakly correlated with the members' average intelligence, and hardly more with that of the brightest member[2]. What predicts it are 4 conditions:

- **the members' social sensitivity** — their ability to read others' mental states;
- **equity in speaking turns** — groups dominated by a few voices reason less well;
- **the proportion of women**, which acts through the first;
- and, added by another branch of the research, **independence of judgment**: as soon as everyone sees the others' estimates before giving their own, the diversity of opinions collapses without any gain in collective accuracy, and the group's confidence rises while its reliability stagnates[3].

Now reread these conditions against a free-software project.

- **Social sensitivity**: the literature documents open incivility in the high places of the free-software world — the Linux

kernel mailing list first among them — but I have experienced circles that were perfectly polite and nonetheless perfectly violent, all the more insidious for it. The point is that the condition does not measure courtesy: it measures the ability to read others' states, exercised for the benefit of the task and of the others. A veneer of benevolence can coexist with its exact inverse — reading others' emotions in order to turn them against them — and the chapter "The closed circle" will take that reversal apart, evidence in hand: politeness set up as a tribunal, tone policing.

- **Speaking-turn equity**: the chapter "Thinking against oneself" counts who merges external code contributions, and the rest of this one will show who decides. There is no equity of speech without a debate facilitator, hence without organization.
- **Gender balance**: one-seventh of the personnel is female, at best, depending on the measurement method — the book's conclusion returns to this.
- **Independence of judgment**: everything gets discussed in sequential public threads, where everyone reads the opinions already given before writing their own — Asch's apparatus deployed at project scale. The opinion of influential people frames the opinion of those who add their comment afterward; the patterns of influence are silent but real.
- **Aggregation**, finally, because you still need a mechanism that converts private judgments into a decision: counting

> "likes" is not one — it converts nothing, it applauds. Failing that, whoever speaks loudest gets heard.

None of these conditions is met. What remains, then, is what numbers do all by themselves, and that is elementary physics. Add dancers to a ball: you only increase the probability of getting your feet stepped on. That is entropy — more particles in random motion in a closed volume, hence more collisions — and to gain efficiency by adding particles, they have to stop moving at random, which gets harder the more of them there are. Collective intelligence is therefore not the sum of individual intelligences: it is a conditional property, which emerges only if you manage to channel individual possibilities into a context that favors them — a context that must be built deliberately.

Collective intelligence must now be put back in its place, and taken out of the cargo cult. The first studies on collective intelligence measured transient groups, on one-hour tasks that demanded little particular skill — nothing that resembles years of distributed software design. Their results therefore transfer only partially to design in an open-source setting. The collective factor and its prediction by social sensitivity hold in groups that collaborate only through written messages, without ever seeing each other — reading the other's state between the lines works like reading it in the eyes[4] — and the great 2021 meta-analysis, on 1,356 groups, confirms the factor and orders its predictors: the collaboration process first, individual skill second, group composition last[5]. The framework thus carries over to our free-software forums. But the same meta-analysis restores what the collective-intelligence slogan spirits away: the individual skill of

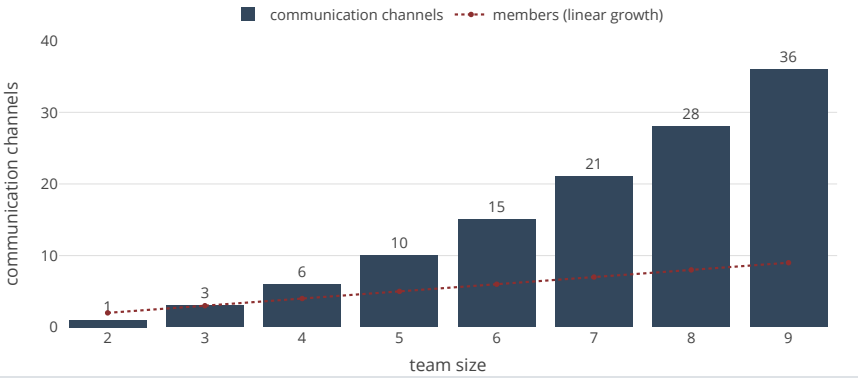
the members is the second factor of collective success — about half as strong as process, admittedly, but its share grows with the technicality of the task. Thus, on a purely technical test like sudoku, it alone explains more than half of the variance[5]. Process makes a competent group reason better; it does not create competence. In this book's vocabulary: training and the trade are not enough to make a collective design well, but without them nothing emerges at all — a necessary condition, never a sufficient one, and therefore the first budget line of any project that claims to design. The common error of collectives — not only in free software, but that is the extreme case where the error costs most, since its product is technical — is to believe that the group can make competence emerge ex nihilo through the sheer force of collaboration. That is not what the literature concludes.

Still, designing is neither estimating nor guessing: it is producing. And on idea-production tasks, the literature is harder still on the committee. We have known since the 1980s that brainstorming groups produce fewer ideas, and no better ones, than the same individuals working separately with their lists pooled afterward; the dominant mechanism is production blocking — only one person speaks at a time, the others wait, and what they were waiting to say evaporates[6]. The meta-analysis confirms the loss and gives it a slope: it grows with group size[7]. This chapter's title is not a provocation, it is an experimental result. And for the precise case of this book — desktop software aimed at end users — the canonical field study of software design says where the scarce resource lives: across 17 large projects, the first of the 3

problems identified is the thin spread of application-domain knowledge, and the fate of projects hinged on one or two exceptional designers, able to build the bridge between the application's expected behavior and the software structures that implement it — often, by their own admission, mediocre programmers, but the keepers of the project's vision[8]. Designing a tool for photographers requires knowing what photography is. That knowledge does not aggregate by ballot, cannot be improvised in a discussion thread, and no group process makes it emerge in those who lack it: the part "The discipline" established this as method, the field confirms it as data.

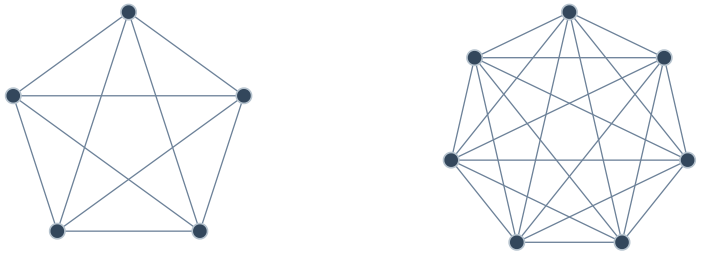
Brooks's arithmetic gives the slope: communication channels grow as $n(n-1)/2$ — 6 channels at 4 people, 28 at 8[9]. Group psychology gives the threshold: work-team performance plateaus between 4 and 7 members; from 8 on, productivity drops, and beyond 9 come cliques, internal politics, and social loafing — individual effort diluted in numbers[10]. And Conway gave the architectural consequence, which explains a good share of the monsters of the part "The machine": a system copies the communication structure of the organization that builds it[11]. A swarm of disconnected individuals, with no division of labor and no prior communication, produces a chaotic architecture: incoherent, redundant, juxtaposed. I saw its cost in pure form, while cleaning up the code Ansel inherited from Darktable: several features had been re-implemented several times, at different moments (sometimes 6 months apart) and in inconsistent ways, so that the same task gave a different result in different places in the software. Two people had solved the same

problem without talking to each other, and the maintainer had not seen the redundancy; Conway's law is not an abstraction, it is that waste, repeated.



The Brooks slope: the number of communication channels, n(n−1)/2, as a function of team size (bars), against the linear growth of headcount (red dotted line). Going from 4 to 8 people doubles the team but more than quadruples the channels — from 6 to 28; at 9 members, every newcomer creates 8 more conversations to maintain.

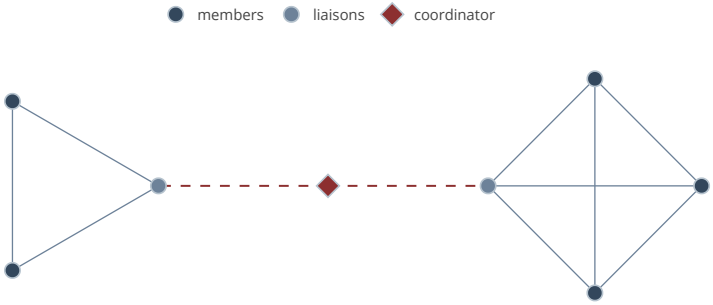
This idea must be pushed to its engineer's form: a team's communication is a network, and that network is an object of design. Draw the graph — people as nodes, communication channels as edges.



The full mesh, drawn out: at 5 people, 10 channels — the network is still readable; at 7, 21 channels — each member holds 6 conversations, and the graph is already denser than any participant can follow. This is the figure of the ball: everyone talks to everyone, and information no longer has a privileged path.

Communication itself is a matter of dosage: it is a means in the service of a goal, not an end. Since the 2010s, and especially in technical circles, the tendency has been to focus the blame on the lack (or even absence) of communication, but too much communication produces exactly the same result: people stop listening to the exhausting stream of information to salvage a minimum of productivity. Not talking to each other and no longer listening to each other end in the same breakdown of the information flow, and they are the two reefs of communication, which must be regulated, directed at a subject, with a place for off-topic matters and a place for deciding. Once consensus is reached on the goals, you move to action: you do not communicate for the pleasure of communicating.

The answer is not less communication, it is *structured* communication: sub-teams of 4 to 7, whose internal exchanges are dense and whose external exchanges go through defined interfaces — a liaison, a written report, a place of decision. The gesture is recognizable: it is Parnas's information hiding, applied to the organization — and this is not an analogy of my own making, it is the conclusion of his 1972 paper, which notes that the structure of the modules and that of the organization end up mirroring each other. Conway then reads in both directions: since the system copies the communication graph, you can *design the communication graph to obtain the architecture you want* — the industry calls this the inverse Conway maneuver, and there now exists a whole engineering of team topologies: which teams, with what boundaries, interacting in what modes[12]. A team that leaves its communication network to improvisation endures its architecture; a team that draws it chooses it.



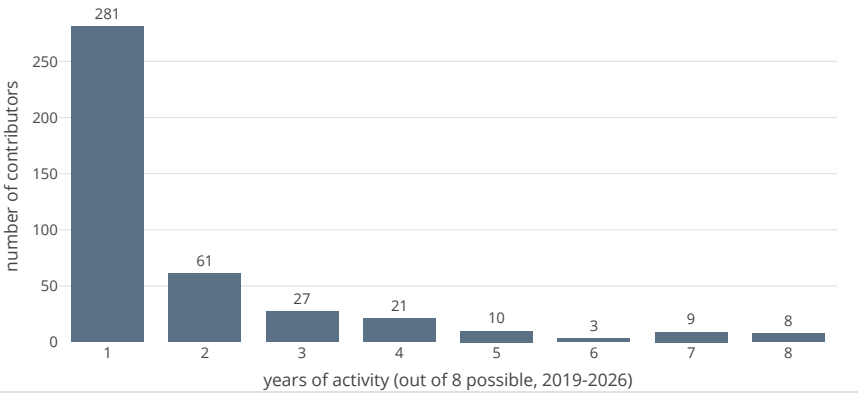
The same network, designed instead of endured: a team of 3 and a team of 4, dense on the inside, communicating with each other through their liaisons (gray) and a coordinator (red diamond). 8 people, 11 channels — 3 in the first team, 6 in the second, two of interface — against 28 in a full mesh: fewer than half the conversations, and every piece of information now has a known path. The price is visible too: the coordinator is a mandatory waypoint — a power, which must therefore be named, bounded, and revocable, failing which you fall back on Freeman.

What does a graph left to improvisation look like? I eventually measured it, and the numbers must be given, for they exceed intuition. On Darktable's GitHub tracker there arrive, on average, day after day and without a single day's pause:

- 6 to 7 new discussion threads — issues (the tracker's term lumps together bug reports and feature requests indiscriminately) and pull requests combined — about 2,400 per year, a volume stable over the last 3 years;
- some twenty discussion messages (updates);

- 3 or 4 line-by-line review comments on pull requests;
- 6 closures or merges of pull requests.

Nearly 40 events per day — one every 25 waking minutes, 365 days a year — and that is a floor, which counts neither the formal reviews, nor the code pushes on watched threads, nor the mentions[13]. It may seem little in absolute terms, but you have to compare it with the personnel that handles the flow to judge it. All of it arrives on a single channel, addressed to everyone — that is, to no one in particular: no triage at the entrance, no separate place for off-topic matters, no interface between the user who reports and the developer who codes. It is the full mesh of the previous paragraph, observed in production. Who absorbs this flow? Over 3 weeks of July 2026, 99 accounts wrote at least one message — and two people alone carried a quarter of the replies. For the population holding the other end of the channel is minuscule, and it is shrinking. Since 2019, 420 people have contributed at least one commit to the project; two-thirds were active only a single year — the median contributor is a passerby; 30 lasted 5 years or more, 8 have been present every year, and 5 authors alone sign more than half of the commits. The chapter "Ship, don't dump" tracked these numbers over time: a peak in 2020-2021, half melted away since — the core that carries the code now fits in a dozen people, active mostly between 5 and 10 pm.



The same 420 contributors, ranked by number of years of activity between 2019 and 2026: two-thirds stay only one year; 8 people are present every year.

That is the full mesh in numbers: a torrent of unsorted, unranked interactions that converges, in practice, on a handful of permanent members whom nothing protects — the chapter "Working for the work" will measure what each interruption costs, and you must imagine it paid 40 times a day by the very people who write the code. A communication graph that no one has drawn is not a free graph: it is a funnel.

This torrent, it must also be said, is a torrent in decline. The chapter "Ship, don't dump" measured it: the inbound flow of issues has fallen by nearly half since 2020, and the assiduous reporters — those who followed up on their reports and filed many — have deserted the counter; hardly anything remains but the passerby's isolated report. Communication does not overflow because it is too abundant, then: it overflows even as it dries up.

And the drying up probably has the same cause as the overflow: a report left to rot unanswered until robotic closure teaches its author only one thing — stop reporting; the lack of hands to process the flow has dried up the source. The project thus suffers from too little communication in general, and from more communication than a team without organization can sustain in particular. The problem was never the volume: it is the absence of any device between the volume and the hands.

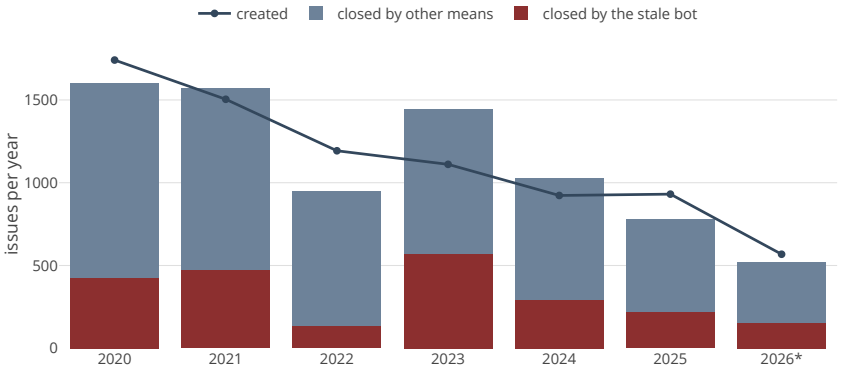
And the funnel has a consequence no one dares state: at this throughput, the record becomes optional. For lack of separate places — one to report a breakdown, one to collect a need, one to deliberate among contributors — everything arrives in the same place and nothing has rank. What has no rank gets read when there is time, that is, never. The IRC/Matrix chat room's log keeps the trace of this, in the most economical form there is. In January 2023, an outside critic takes the project to task over its interface flaws, contrast measurements in hand, and mentions the issue he has filed; no one had seen it, to which a contributor concludes: "if you don't read the issues, then there are no issues"[14]. The sentence is a joke, and that is what makes it damning: the mechanism is familiar enough in that milieu to make an insiders' joke.

The main channel, for its part, loses information by construction: IRC keeps no history. A permanent conversation without a consultable archive informs only those who were present at the right moment: "reading backlog is bad for your health," the founder summed up in 2022, while I had been arguing for 4 years that this channel is unfair to anyone not permanently connected

to it[15] — all the more so once you account for everyone's time zones, in an international project. A project that deliberates where nothing can be reread does not decide less: it decides without memory and without witnesses. It is the communicational version of Freeman's verdict — not too much structure, but none, and the power that lodges there — a verdict this chapter will quote in full once its measurements are laid down.

What the funnel does to the work can be read in the records, for the Git version-control system on which GitHub is built keeps every archive, over which one can easily script systematic statistical extractions at the scale of the project's lifetime. The processing regime for bug reports is *now or never*: over the last 4 months, half of the closed issues were closed within a median day and a half — the responsiveness is real — but 47% remain open, and the net balance grows by about 170 issues per year; there are today more than 500 issues waiting, about forty per member of the working core. Triage no longer exists: 44% of recent issues carry no label at all, only 6% have been confirmed by reproduction — and the most frequent label in the record is the one affixed by the *stale bot*, the automaton tasked with noting that a thread has gone unanswered. The statistically most likely fate of a recent issue, after "closed within the day," is to be declared dead by a machine. And this is no accident of the sample: since the tracker opened in 2019, about 9,000 issues have been filed, and nearly one in 3 has ended up staled by that bot; year in, year out, the automaton signs 3 closures out of 10. The only actor that empties the record at scale is not a human who

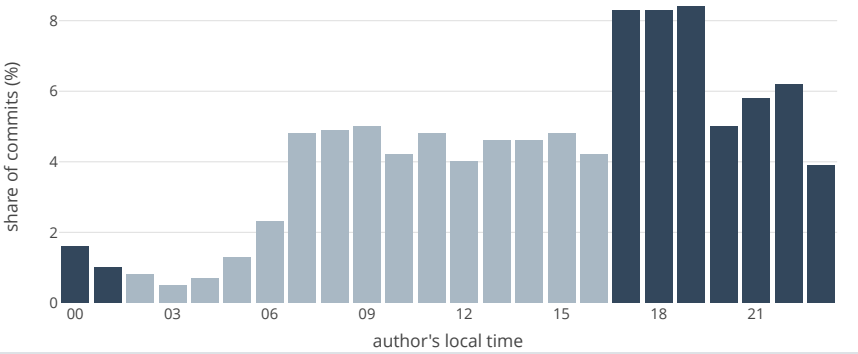
answers: it is an inactivity counter. And do not think the inside is unaware of it: "nobody looks at gh anyway," a core contributor wrote in the project's chat room as early as summer 2022; and in late 2024, when the software's Debian packager objected that he "would prefer that non-bugs are closed with an explanation and that open bugs stay open… bugs accumulating is the lesser evil," the bot's operator answered with a question that sums up the whole regime: "Would you prefer that such issues never get closed due to age and their number grows?"[16] Closure by age and by volume is not a deplored accident: it is the policy, defended as such by the man who executes it.



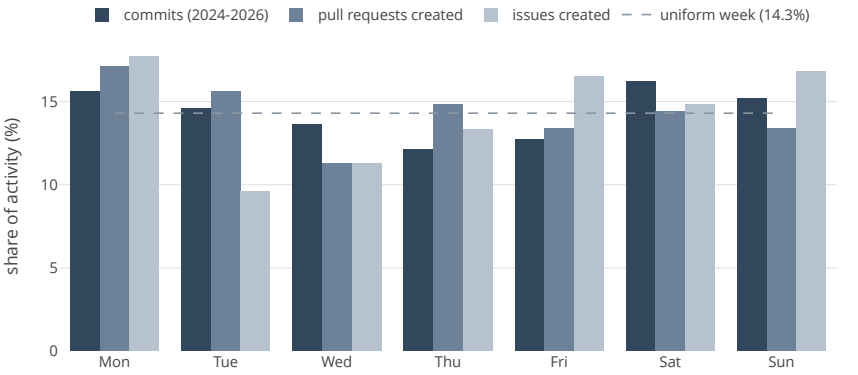
The end of life of Darktable issues, by year of closure: the red share is closed by the stale bot — the automated finding that no answer ever came — and the line gives the issues created. The record only empties durably with the bot's help; 2026 stops at July 22.

Pull requests, for their part, get merged in a median 10 hours — a speed that forces a question the chapter "The cargo cult" settled

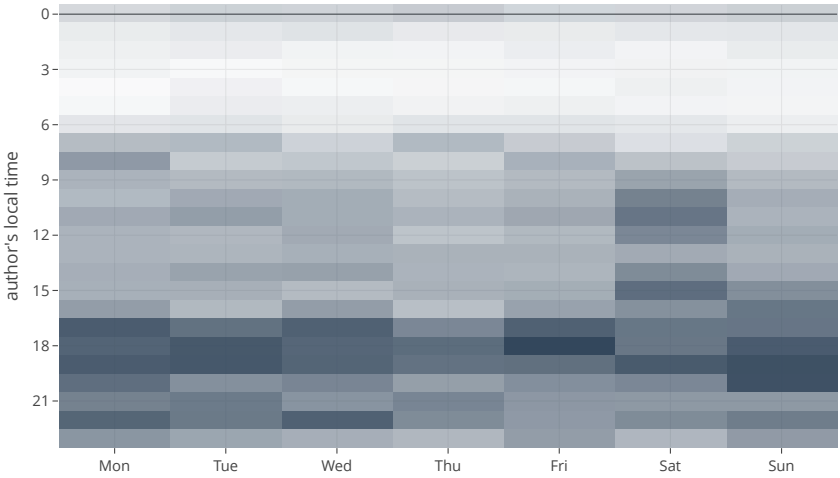
by arithmetic: at this throughput, on this volume, with this team — and a single man holding merge rights — the possible depth of review is zero. As for the project's heartbeat, it has none: the inbound flow of reports and pull requests is flat across the week — users report on Sunday as on Tuesday — and the commits are nearly as flat, barely swollen on weekends; it is the hour of the day that tells everything, for the work concentrates between 5 pm and midnight, each author's local time — 40% of commits after 6 pm. People who hold jobs elsewhere pour their evenings, every evening, into a flow that knows neither week nor respite[17]. The system holds — but it holds by renouncing precisely what this book calls designing: triaging, confirming, prioritizing, periodizing. The hypothesis is not that these people work badly; it is that this volume, on this graph, is unmanageable by this headcount, and that the whole organization has reconfigured itself around that impossibility, abandoning everything that does not make enough noise, but also emptying the room through discouragement.



The project's heartbeat: hourly distribution of the 5,652 commits of 2024-2026 (merges excluded), in each author's local time. The peak is in the evening — 40% of commits after 6 pm: employees' evenings, poured into a flow that never stops.



The week that does not exist: distribution by day of the week of commits (2024-2026), pull requests, and issues created (March-July 2026 sample). Everything stays within a few points of the uniform week: neither users nor developers have a weekly rhythm — the flow knows no rest.



The heat map of the work, 2016-2026: 26,865 commits broken down by day of the week and author's local hour. The dark patch covers the evenings of every day of the week — the project lives after office hours, 7 days a week.

Then comes the argument that serves as a constitution for many projects: no boss, no hierarchy, no structure — freedom. This is where the most important quotation of this book on matters of governance belongs, and it comes from the American feminist movement, which ran the experiment before software did. In 1972, Jo Freeman published "The Tyranny of Structurelessness": the absence of structure does not exist[18]. A group that refuses all formal structure does not remove power; it makes it invisible. Informal elites form there — those who were there before, who know each other, who hold the channels — and they wield a power all the more unaccountable in that, officially, it does not exist: you can neither elect, nor evaluate, nor depose what has no

name. Only explicit structure makes power accountable, precisely because it makes it visible. And Freeman added the operational corollary: structurelessness cannot direct work. It is all there. The project proud of having neither hierarchy nor procedure has one nonetheless — the same individuals locking the same decisions for years, without mandate, without evaluation, without counter-power — and this book's prologue said what it costs when public pressure is the only lever there is. The problem is not that people decide; it is that they decide without being either appointed or accountable, and that the system sheltering them makes that lock possible.

What do we put in its place, if neither the committee nor structurelessness directs work? Leadership — the word is worn out, so it must be defined by its 3 properties. It is *earned*: research on free-software communities shows that legitimacy there is built over time and granted conditionally — neither founding the project nor technical talent confers it on its own; the leaders who last do organizational work, not just code[19]. It is *accountable*: structured, visible, evaluable — exactly what Freeman opposes to informal elites; authority is bounded by explicit mechanisms, not by good will. And it is *capable of saying no*: the failure mode of Darktable's maintainer lies there, in the gap between the code written and the code accepted — accepting, to be nice, code you will not be able to maintain, dodging the hard decisions to preserve the mood, is overloading yourself and bequeathing the problem to the project. There remains the symmetric risk, for a leadership that says no soon surrounds itself with silence: groupthink, described by Janis — cohesion smothering

disagreement all the way to disaster[20]. The countermeasures can be organized: a culture where dissent is a service rendered and not an attack, anonymous votes when a subject heats up, a rotating devil's advocate; and research on teams has named the condition of possibility of all this, psychological safety — problems get reported only where reporting does not cost[21]. As for the majority vote, it remains a tool of closure, not a tool of design, and it mechanically oppresses structural minorities — the chapter "Counting the absent" showed who is missing from the samples; the same arithmetic holds in ballots. The dividing line this book will defend to the end is therefore this: the collective is sovereign over the *goals* and the budgets — over what deserves to be solved; the *way* of solving remains a trade, with a person in charge who has a name.

The objection has a founding text, the most influential in the history of free software. In "The Cathedral and the Bazaar," Eric Raymond opposes cathedral development — planned, hierarchical, closed — with the bazaar: everyone scratches their personal itch, you release early and often, and "given enough eyeballs, all bugs are shallow"[22]. The text convinced a generation that structure was the problem and the crowd the solution. Four answers. First, the bazaar described in the text was never structureless: the Linux kernel, its central example, has always had a maintainer at the top and a hierarchy of lieutenants — a cathedral that accepts the bazaar's deliveries. Second, the many eyes find the shallow bugs, literally: the metadata overwriting of the chapter "Measuring without lying to yourself", the 18 requests per hover of the chapter "Beneath the interface,

dragons" sat for years under thousands of eyes, because an architecture cannot be reviewed in passing — depth is not accessible to the distributed gaze. Further, the personal itch as an engine produces exactly the accumulation the chapter "In praise of the simple" described: features exciting to write, never maintenance, boring by definition. Finally, the Linux kernel is not a line-of-business application with a graphical interface for end users, but the core of an operating system that remains invisible to users' eyes: how many Android users know it is based on Linux? Interacting with Linux is reserved for application developers alone, through configuration files and protocols, and no user "uses Linux": what people actually use are distributions based on Linux.

Two questions separate the contributor from the designer here: driven by desire, the first asks "what if we added…?"; driven by the problem, the second asks "what is missing, for whom, and how do we remedy it?" The first question adds up features; only the second designs. But the best answer to the bazaar lies in its own data: the largest study ever conducted on the fate of free-software projects — 174,000 projects tracked — concludes that success is predicted by *leadership by example*, a clear vision, and clearly stated goals[23]. The romanticism of the bazaar has survived as a slogan; its own data vindicate the opposite model — a project led by someone who knows where they are going.

The committee will invent nothing, structurelessness will direct nothing: in both cases, the work falls back on someone — the same someone, usually, the one who maintains, documents, triages, and answers while the discussion goes on. It is time to

look at this worker whom everyone uses and no one sees. That is the subject of the next chapter.

---

1. Asch, S. E. (1951). "Effects of Group Pressure upon the Modification and Distortion of Judgments," in H. Guetzkow (ed.), *Groups, Leadership and Men*. Pittsburgh: Carnegie Press, pp. 177-190; popular account: Asch, S. E. (1955). "Opinions and Social Pressure." *Scientific American*, 193(5), 31-35 https://doi.org/10.1038/scientificamerican1155-31 ; monograph: *Psychological Monographs*, 70(9, no. 416), 1956 https://doi.org/10.1037/h0093718 . Figures: about one-third of the answers on critical trials follow the unanimous majority into error (less than 1% errors in the control condition); about 75% of subjects yield at least once — and about 25% never yield: Asch himself insisted on independence as much as on conformity. A single dissenting confederate brings conformity down to around 5%: it is unanimity that makes the power, not the number — hence the value, in the chapter "Thinking against oneself", of the instituted dissenter. ↩

2. Woolley, A. W., Chabris, C. F., Pentland, A., Hashmi, N. and Malone, T. W. (2010). "Evidence for a Collective Intelligence Factor in the Performance of Human Groups." *Science*, 330(6004), 686-688. https://doi.org/10.1126/science.1193147 Two studies, 699 participants, groups of two to 5: a general factor explains 43-44% of the variance in group performance. Correlations with member intelligence: r = 0.15 (average) and 0.19 (maximum) — significant but weak. Predictors: social sensitivity, measured by the "Reading the Mind in the

Eyes" test (r = 0.26); variance in speaking turns (r = −0.41, sociometric badges, on a subset of the groups); proportion of women (r = 0.23, effect mediated by social sensitivity). ↩

3. Lorenz, J., Rauhut, H., Schweitzer, F. and Helbing, D. (2011). "How Social Influence Can Undermine the Wisdom of Crowd Effect." *PNAS*, 108(22), 9020-9025 https://doi.org/10.1073/pnas.1008636108 — "social influence effect," "range reduction effect," "confidence effect": the 3 terms are the authors'. The canonical formulation of the conditions for the wisdom of crowds — diversity of opinion, independence, decentralization, and an aggregation mechanism turning private judgments into a collective decision — is from Surowiecki, J. (2004). *The Wisdom of Crowds*. New York: Doubleday, chap. 1. https://openlibrary.org/books/OL7440777M/The_Wisdom_of_Crowds ↩

4. Engel, D., Woolley, A. W., Jing, L. X., Chabris, C. F. and Malone, T. W. (2014). "Reading the Mind in the Eyes or Reading between the Lines? Theory of Mind Predicts Collective Intelligence Equally Well Online and Face-To-Face." *PLoS ONE*, 9(12), e115212 https://doi.org/10.1371/journal.pone.0115212 — 68 groups of 4, of which 36 collaborated only through written messaging: the collective factor explains 41% of the variance online (49% face-to-face), and the mental-state reading test predicts performance even when no one sees anyone. ↩

5. Riedl, C., Kim, Y. J., Gupta, P., Malone, T. W. and Woolley, A. W. (2021). "Quantifying Collective Intelligence in Human

Groups." *PNAS*, 118(21), e2005737118 https://doi.org/10.1073/pnas.2005737118 (correction: *PNAS*, 119(19), 2022, with no effect on the conclusions) — 22 studies, 1,356 groups, 5,279 individuals. In their terms: the collaboration process predicts collective intelligence more than member skill does; skill comes second, its share depending on the task (more than 51% of the variance explained on sudoku); the proportion of women remains a significant predictor, mediated by social sensitivity. ↩ ↩

6. Diehl, M. and Stroebe, W. (1987). "Productivity Loss in Brainstorming Groups: Toward the Solution of a Riddle." *Journal of Personality and Social Psychology*, 53(3), 497-509 https://doi.org/10.1037/0022-3514.53.3.497 — of 22 earlier experiments surveyed, 18 favor nominal groups and none favor real groups, in quantity as in number of good ideas; their 4th experiment isolates production blocking as the main mechanism, ahead of evaluation apprehension and free riding. ↩

7. Mullen, B., Johnson, C. and Salas, E. (1991). "Productivity Loss in Brainstorming Groups: A Meta-Analytic Integration." *Basic and Applied Social Psychology*, 12(1), 3-23 https://doi.org/10.1207/s15324834basp1201_1 — a loss "highly significant and of strong magnitude", in quantity as in quality, correlated with group size (r = 0.61 for quantity, 0.72 for quality); the authors favor social mechanisms over blocking alone — the disagreement concerns the mechanism, not the loss. ↩

8. Curtis, B., Krasner, H. and Iscoe, N. (1988). "A Field Study of the Software Design Process for Large Systems." *Communications of the ACM*, 31(11), 1268-1287 https://doi.org/10.1145/50087.50089 — 17 large projects, 9 companies, 97 interviews. First of the 3 problems identified: "the thin spread of application domain knowledge." The exceptional designers, one or two per project, extremely familiar with the application domain and described as those who saved the system, often called themselves mediocre programmers and "were recognized as the intellectual core of the project (i.e., the keeper of the project vision)". ↩

9. Brooks, F. P. (1975). *The Mythical Man-Month*. Reading, MA: Addison-Wesley (1995 anniversary ed.) https://openlibrary.org/books/OL5044740M/The_mythical_man-month — the arithmetic of intercommunication channels, n(n−1)/2, and the law that bears his name: "Adding manpower to a late software project makes it later." ↩

10. Hackman, J. R. (2002). *Leading Teams: Setting the Stage for Great Performances*. Boston: Harvard Business School Press https://openlibrary.org/books/OL8754168M/Leading_Teams; Wheelan, S. A. (2009). "Group Size, Group Development, and Group Productivity." *Small Group Research*, 40(2), 247-262 https://doi.org/10.1177/1046496408328703; Allen, N. J. & Hecht, T. D. (2004). "The 'romance of teams'." *Journal of Occupational and Organizational Psychology*, 77(4), 439-461 https://doi.org/10.1348/0963179042596469. On social loafing: Latané, B., Williams, K. & Harkins, S. (1979). "Many Hands Make Light the Work." *Journal of Personality and Social*

Psychology, 37(6), 822-832 https://doi.org/10.1037/0022-3514.37.6.822 . ↩

11. Conway, M. E. (1968). "How Do Committees Invent?" *Datamation*, 14(5), 28-31 https://www.melconway.com/Home/Committees_Paper.html : "organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations." The title of this chapter echoes his. ↩

12. Parnas, D. L. (1972), *op. cit.* https://doi.org/10.1145/361598.361623 — the correspondence between the structure of the modules and that of the organization producing them is sketched in the original paper. For contemporary practice: Skelton, M. & Pais, M. (2019). *Team Topologies: Organizing Business and Technology Teams for Fast Flow*. Portland: IT Revolution Press https://books.google.com/books/about/Team_Topologies.html?id=Pj-IDwAAQBAJ — team types, interaction modes, and the "inverse Conway maneuver" (designing the organization to induce the target architecture). ↩

13. Counts made on July 22, 2026 on the https://github.com/Darktable-org/Darktable repository via the GitHub API. New threads: 2,256, 1,946, then 2,400 issues and pull requests created over the July-June windows 2023-2024, 2024-2025, and 2025-2026. Messages: 473 comments from July 1 to 22, 2026 (21.5 per day, of which 2.6 issued by the continuous-integration bot); line-by-line reviews: 3.4 per day over the same window; closures and merges: 2,196 over

July 2025-June 2026. Contributors: full git history, merges excluded, authors deduplicated by name, from January 1, 2019 to July 22, 2026 — 2026 is a partial year of six and a half months. ↩

14. Log of the project's public Matrix room (bridge of the IRC channel), July 23, 2026 export, exchange of January 18, 2023 between 12:15 and 12:20 pm. The founder posts two actions there: "*doesn't follow Darktable development so closely*" and "*goes read the github issue*"; a contributor follows with "if you don't read the issues, then there are no issues," then jokes about not logging into his own employer's ticketing system. Register: it is self-deprecation, not a factual report — I quote it as an aphorism that names a mechanism, not as testimony. ↩

15. Same log, March 25, 2022. The founder: "reading backlog is bad for your health"; and, the same day, my own remark: "I have spent the past 4 years trying to convince ppl that IRC is incredibly unfair to those who don't live on the same time zone as most of us, aka western Europe, because there is no backlog". The "I haven't read the backlog" motif is ubiquitous among the room's regulars that same year. ↩

16. Log of the project's public Matrix room (July 23, 2026 export, see the chapter "The closed circle"): "nobody looks at gh anyway," July 2, 2022; exchange of December 1, 2024 — the Debian packager: "I would prefer that non-bugs are closed with an explanation and that open bugs stay open… bugs accumulating is the lesser evil"; the bot's maintainer:

"Would you prefer that such issues never get closed due to age and their number grows?" ↩

17. Same cutoff date, same sources. Processing sample: the last 1,000 threads (March 19-July 22, 2026; 345 issues, 655 pull requests). Latencies: median closure of 1.3 days on closed issues (53%), 90th percentile 22 days; pull requests merged in a median 0.4 days. Labeling: 44% unlabeled; "no-issue-activity" (the stale bot) = 55 occurrences, the record's most frequent label; "reproduce: confirmed" = 19. Backlog as of July 22, 2026: 535 issues and 84 pull requests open; net balance over 12 months: +167 issues. The issue series start in 2019 because the project migrated its bug tracking from a self-hosted Redmine to GitHub in late 2018 (the old tracker has redirected since December 2019) — the boundary is the archive's, not a choice. Heartbeat: weekly distribution of thread creation flat (9.6 to 17.7% per day); commits 2024-2026 (n = 5,652, merges excluded): 31.5% on weekends, 40.1% between 6 pm and 2 am, author's local time as recorded by git. Staleness: 2,589 issues carry the inactivity bot's label, out of 9,102 created from 2018 to July 22, 2026 (28%); labeled closures per year, 2020 through 2026: 424, 476, 132, 574, 297, 222, 153 — that is, 27 to 40% of each year's closures (first labeled closures observed in 2020). Heat map: commits excluding merges, 2016-2026 (n = 26,865), author's local time. The headcount and departure series and the monthly pulse are in the chapter "Ship, don't dump". ↩

18. Freeman, J. (1972). "The Tyranny of Structurelessness." *Berkeley Journal of Sociology*, 17, 151-164

https://www.jofreeman.com/joreen/tyranny.htm — written from the experience of the women's liberation movement; the text has circulated ever since as a pamphlet in activist movements, where the lesson gets relearned by each generation. ↩

19. O'Mahony, S. & Ferraro, F. (2007). "The Emergence of Governance in an Open Source Community." *Academy of Management Journal*, 50(5), 1079-1106 https://doi.org/10.5465/amj.2007.27169153 — a longitudinal study of Debian: legitimacy there is built and granted over time, and the community built a formal authority and then bounded it with democratic mechanisms. ↩

20. Janis, I. L. (1972). *Victims of Groupthink*. Boston: Houghton Mifflin. https://openlibrary.org/books/OL19773076M/Victims_of_groupthink The mechanism requires cohesion and the pursuit of unanimity — a committee without cohesion produces noise, not groupthink; the two pathologies frame the same absence of method. ↩

21. Edmondson, A. (1999). "Psychological Safety and Learning Behavior in Work Teams." *Administrative Science Quarterly*, 44(2), 350-383 https://doi.org/10.2307/2666999 . ↩

22. Raymond, E. S. (1999). *The Cathedral and the Bazaar*. Sebastopol: O'Reilly (initial essay 1997) https://www.catb.org/~esr/writings/cathedral-bazaar/ . Lesson 1: "Every good work of software starts by scratching a developer's personal itch"; lesson 7: "Release early. Release

often.”; lesson 8: “Given enough eyeballs, all bugs are shallow” (so-called Linus’s law). ↩

23. Schweik, C. M. & English, R. C. (2012). *Internet Success: A Study of Open-Source Software Commons*. Cambridge, MA: MIT Press https://direct.mit.edu/books/monograph/3364/Internet-SuccessA-Study-of-Open-Source-Software — 174,333 SourceForge projects and 1,403 developers surveyed: the predictors of success are leadership by example, a clear vision, and clearly stated goals; the governance of successful projects is light and informal at first, formal institutions mattering only later. ↩

21. The invisible worker

Throughout this book, we have thought about design in the service of the user. This chapter turns the lens around, and it states the thesis that took me the most years: **the quality of life of the one who does the work is a design criterion too.** It is the great blind spot. We measure everything — clicks, times, distributions — and we ignore how painful it can be simply to get the work done. It took me a long time to see it, out of a technician's bias: my engineering training gave outsized weight to the technical side of things. Yet the work environment, the way of working and the comfort of working matter at least as much as the intelligence deployed in finding good solutions — because behind every technical problem, there is a human, including the one who writes the software. The phrase opens and closes this book; this chapter takes it seriously.

It has to be stated right away, because everything that follows hinges on it: the discipline described in these pages is a discipline of the care owed to others, and a care that exhausts the one who gives it is not a poorly rewarded virtue — it is one more exploitation, with a clearer conscience. Besides, working conditions are also a factor of attraction or repulsion for the workforce, and caring for the worker is how you avoid watching your ranks melt away over time, as we saw in the previous chapters with Darktable.

To make the worker visible, you first have to make the work visible, for the public sees only the tip of it: the commits, the bustle of the repository. Here is the actual inventory of my own case:

- **development**, plus reviewing other people's code;
- **documentation**: dozens of pages, maintained in two languages;
- **teaching**: dozens of hours of video tutorials, to show which module is for what, in what context, with what benefit;
- **support**: the forums, and every channel where people can find you;
- **stewardship**: triaging bug reports, packaging releases, the build infrastructure, moderation;
- **and the flow of communication that wraps around it all**: for the year 2021 alone, GitHub sent me more than 4,000 emails; over the first 8 months of 2022, the community forum added nearly 700; and I was being solicited in parallel on YouTube, Reddit, Matrix, Telegram and by direct email.

Outside foundation-backed projects, this level of support and documentation simply does not exist in free software — and where it does exist, it is generally not the developers who carry it[1]. I carried both, having become the public face of the project without ever deciding to, and with no staff to screen the solicitations. The economics of the whole thing fits in two numbers: I never counted more than 240 donors, while my two surveys gathered more than 2,000 respondents, whose median declared photography spending was around 500 euros a year.

People only respect work once it has been billed to them at full price.

The first criterion of comfort is a work regime, and the chapter "Finishing and maintaining" laid out its mechanics: work must capitalize. The regime to flee at all costs is the one where the more you work, the more you have to work — where fixing one defect creates another, where everything is bought with technical debt, where nothing accrues. Working well means accumulating: tools, methods, understanding. Every hour spent equipping yourself with a diagnostic tool, a clean collection of bug reports, a test that locks in a gain, is an hour that then repays itself indefinitely; you plug the holes as you go, and the edifice becomes watertight. The more you work, the less you *need* to work. And the best way to lighten the work is not always to write code: a documentation page that dries up a source of questions, a short video, a reorganized menu do the same job. The rule of thumb is simple: think from the point of view of whoever will be running the machine 6 months from now — most often yourself. Without method, without tools, without a shared goal or a shared task list, a team of 10 or 15 people finds itself on quicksand, absorbing one surprise after another: always reacting, never acting, let alone thinking. Conversely, the situations where work breeds more work are the telltale sign of infernal machines in full runaway, and it is the work method you must act on before it is too late.

The second criterion is control over exposure, and the asymmetry must be described first, because it is invisible from the outside. In a company, development happens under cover: when you want

user feedback, you organize a panel — a few people in front of the tool, behind a one-way mirror, at a chosen time, with chosen people. You keep control of the timing. In free software, everything is public, all the time; you are permanently exposed, and never at a moment you would have chosen. And this loss of chosen timing cuts both ways: you do not choose when the world's feedback arrives — we will see below what it weighs — but neither do you choose when to stop keeping watch, for everything is decided in public, at all hours, and stepping out of the flow means letting the project decide without you. Two lived scenes give the texture of this servitude of presence. I spend a month in my cave developing a perceptual color space, a long and demanding research project; I emerge to discover that a labyrinthine new image-filtering system has been merged in my absence, and to be told that I am too late to object. I take 3 weeks of vacation in August; on my return, my review of a mathematical change to that same color space has been bypassed — things had to move fast. For what emergency, exactly? In this regime, being away is not resting: it is falling behind on decisions already made. Add to this the asymmetry of statuses: for your users, it is a hobby — evenings, weekends; for you, it is work. So the feedback arrives when it suits them, in the middle of (more) important things, and you have to read everything to triage it, then explain, justify, defend every choice. Multiply by a crowd: you are facing, alone, more cumulative time and energy than you will ever possess. Financial donations do not lighten this load, they weigh it down with expectations — and a longitudinal study of donation platforms confirms their structural impotence: a significant effect on short-term activity, none on the long term;

money without structure buys expectations, not sustainability[2]. Research has given this economy a name: the maintainer's attention is an overexploited common resource — everyone draws on it as if it were free and infinite, and no one is accountable for its depletion[3]. It has also documented the mechanism that turns this exposure into exhaustion: toxic interactions — aggression, contempt, demands without gratitude — are identified as a direct driver of maintainer burnout[4]. The ailment is of its time: creators used to be protected by intermediaries — an agent, a publisher — who filtered the mail and absorbed the blows; today you publish your work directly, a comment box bolted underneath, and you are exposed by default. The community is, moreover, almost exclusively male, and exchanges there often take the form of a power struggle: you have to defend yourself at all times — the chapter "What insularity produces" will say who makes up this community and why it matters.

In this flow, one category deserves its own paragraph: ideas. Users have plenty to spare, but lack the skills to turn them into prototypes. One thing must be understood about ideas: they are worth nothing; what costs is making them real. Nine of your own ideas out of 10 are bad, and you only learn it by trying to build them; whoever lacks the means to realize his ideas never gets that reality check. Convinced that having had an idea makes him someone exceptional, he hands you the idea *and* the job of making his dream come true. An idea costs nothing; delivering a finished product, usable in production — not a half-working prototype — costs everything. The feedback triage of the chapter

"Listening without obeying" here takes on its protective function: the testimony of use gets processed, the delegated fantasy gets politely declined — and the numbers of the chapter "Counting the absent" serve as a shield too, for sending a noisy demand back to the reality of its 5% is a way of buying yourself peace, of protecting yourself.

Then comes the only rule I hold to be vital, learned from the mistake I wish on no one: do not be too available — it is the open door to exhaustion. Availability is a finite resource; it must be treated as such. You decide when you read the feedback, not the people who send it: batch it, filter it, do not answer on the spot, treat no message as an emergency until it is one. Deferring the handling of problems is not laziness; it is quite the opposite. When you change a critical piece of the code, you are fairly sure to break something; the question is: what? A single bug report is rarely enough to trace back to the source of the problem. Leaving the bug open for a while gives time for other bug reports to arrive, which will find problems surfacing in other places but pointing to the same origin. You then get a cluster of bugs that see the same problem from different angles and, by a kind of triangulation, often let you trace back to the origin far more easily. Provided you read them all…

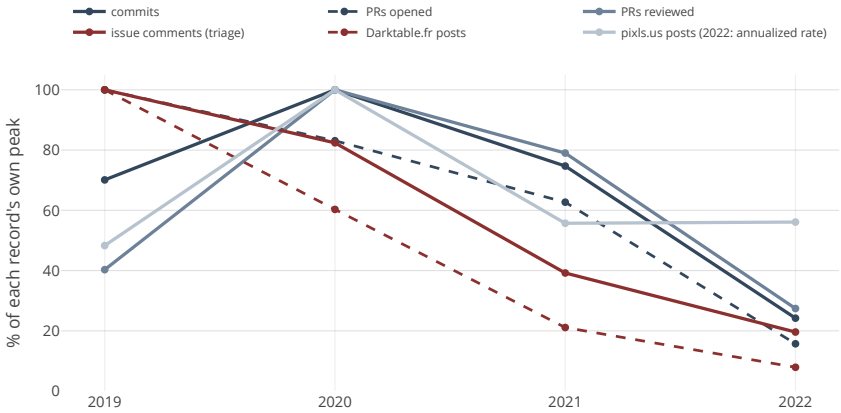
The other argument for deferral is that design work also has a rhythm of its own, which the workshop tradition calls the maker's schedule: creative and analytical tasks exist only in blocks of several hours, and a single ill-placed interruption destroys the entire block[5] — permanent urgency is therefore not merely unpleasant, it is incompatible with the nature of the work.

Nothing good gets designed in a state of permanent emergency; giving yourself margin is not a luxury, it is a condition of design.

Protection can go as far as building tools — the principle here, the full story in the next chapter: not every problem is solved with code, but some communication problems are. In 2022, the volume of solicitations had reached the point of no return: the majority of human messages reduced me to a flesh-and-blood search engine, summoned to retrieve from my own documentation content I had already written, sometimes in both languages. I ended up building the machine that was missing — a triage secretary, then a search engine over my own content; the chapter "Working for the work" tells how. Only the doctrine matters here, and it is that of the chapter "Finishing and maintaining" extended to communication: an hour spent tooling up the filtering repays itself indefinitely; an hour spent answering a documented question for the umpteenth time is lost for everyone — including the one who asked it, for a tired answer is a bad answer.

The cost must also be given its units, for "exhaustion" remains abstract as long as you do not count it out in a body and a calendar. My first carpal tunnel syndrome dates back more than 10 years: typing and handling a mouse is physical work, and the repetitive strain injury of the chapter "Stating the problem" does not spare the one who builds the tool. Software-induced depression appears, under that very name, in my dev diaries. And the great cleanup of the photo library (lighttable) — nearly 300 commits over 4 years to undo workarounds all of which dated from after 2020 — cost me 4 years of my life, without any pleasure. And the exhaustion itself can be read in the

contribution records, with a precision that surprised even me when I measured it. The order of the abandonments is the signature: in 2021, my code holds — 210 commits, an almost normal year — but my interaction contributions are already cut in half: issue triage, the forums, everything that exposes. You trim the contact surface first, to protect the craft. In 2022, the craft gives way in turn — commits, pull requests, reviews: down by two-thirds across the board — with one last flare-up in May, spent in the filter conflict, while my messages in the chat room were exploding. Then the cliff: one commit in July, last visit to the French-speaking forum on the 24th, silence in September[6]. I am, measurably, one of the 2022 data points of my own scissors chart — a profile at full throttle then nothing, not a gradual decline.



Exhaustion, measured in the contribution records, 2019-2022 — each record as a percentage of its own peak. In blue and gray, the craft: commits, pull requests, reviews. In red, the exposure surface: issue triage and the French-speaking forum. The order of the abandonments is the signature: the exposure records are cut in half as early as 2021, a year before the code gives way — what exposes is trimmed first to protect what produces. The English-speaking forum (light gray, 2022 annualized over the first half) holds until the exit: it is the teaching channel, abandoned last. Contribution measures, not a clinical diagnosis; sources in the note.

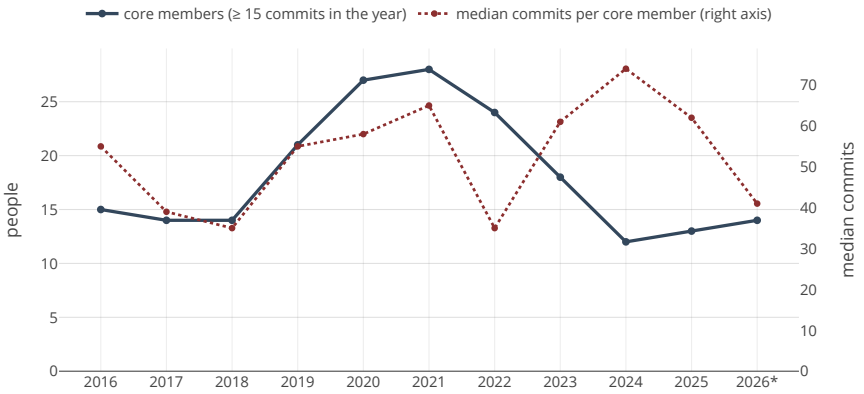
There is an enormous difference between lacking the time to do things well and consuming hours of human labor making things worse; those who introduce the regressions that others painstakingly repair bear the responsibility for the consequences. This is accounting: the repairers' hours are the budget of the chapter "Finishing and maintaining", and someone spends them.

The project's records make it possible to track these exits at scale, and their *shape* tells what the averages hide. I reconstructed the end-of-run profile of the 22 core members — 15 commits or more in their last active year — who have left Darktable since 2016. Only three fade out on a gentle slope, the gradual weakening usually attributed to exhaustion. All the others leave at once, and you have to look at a few of these electrocardiograms to understand what "at once" means:

- **a pillar of the first decade** — 11 years of presence, more than 2,000 commits: his last 3 full quarters *accelerate*, 10, 18, 27 commits, then the record stops dead — two residual commits, and never anything again;
- **a meteor** — 3 years, 800 commits, a pace of 150 to 170 commits per quarter: still 157 in his second-to-last quarter, 9 in the last, then silence;
- **a 5-year member** — one last spike of about a hundred commits in the first quarter of 2022, two residual quarters of 15 then 3 commits, and the exit in the summer of 2022;
- **the doyen** — 14 years of presence, nearly 1,400 commits: his last two years are no more than a flicker — a quarter at 19 commits in late 2023, a year of near-silence, one last beat at 19 in early 2025, then nothing.

To which add some ten flare-ups: almost nothing, one or two intense quarters — 20 commits at once from someone never seen before — then the disappearance[7]. The lesson is not psychological, it is structural: there is no part-time position in such a project. You are core or you are nothing; unable to slow down, you break. And meanwhile, those who stay absorb the

load of those who left: the core melted by half between 2021 and 2024, and the median commits per survivor more than doubled over the same period — 35 in 2022, 74 in 2024, the highest level of the decade[7]. Fewer hands, more load per hand, and no intermediate regime between full blaze and exit: this system does not produce tired people — it produces ruptures. And the quality of the code produced inevitably suffers: code output that doubles per developer necessarily has a price.



The scissors: Darktable's core headcount (≥ 15 commits in the year) against median commits per core member. When the headcount melts by half (2021-2024), the survivor's median load doubles — the 2022 trough is the year of the conflict, when even the output of those present collapses. Commits excluding merges, deduplicated authors; 2026 is partial.

And what these ruptures carry away must be counted too, for it can be read in the files: 39% of the project's source files have by

now lost their principal author — the one who wrote most of the file no longer commits — and 6 people alone hold majority authorship of half the code base[8]. A dead stop does not carry away just a worker: it carries away the knowledge of the files he alone was carrying — with no handover, since leaving at once is precisely leaving without a handover, and since the project keeps no developer documentation (which would, for that matter, add still more to the workload).

Finally, it must be said what burnout is, and what it is not. It is not an individual weakness: it is the normal output of the system described in this chapter — permanent exposure, asymmetry of statuses, extraction of attention with no accounting, absence of filters — applied to an ordinary human for long enough. The numbers of the chapter "Finishing and maintaining" give its prevalence; this chapter gives its mechanism. And since 2024, the argument has changed category. The backdoor discovered in xz-utils — a compression library present in more or less every Linux system in the world — was not inserted through a technical exploit: it was inserted through social engineering, on a solitary and exhausted maintainer, harassed for years by fake accounts pressing him to hand over maintenance to his future attacker[9]. The exhaustion of the invisible worker is no longer merely a matter of well-being: it is an attack surface of the global software supply chain. The maintainer's comfort is a security requirement.

The objection is nothing hypothetical: it was put to me publicly, for years, notably by a moderator of the community forum — "nobody is forcing him to stay" when I described the conditions; and, when my withdrawal for exhaustion was announced, the

victim was sent back to his supposed privilege: “He talks about having too much free time, but who had almost six years to mostly write code?”[10]. The chemically pure form of the argument was given elsewhere, in reply to someone who recalled that you do not push people into burnout without consequences: “Who pushed him? He pushed himself. It was all volunteer work, it was all things he wanted to do. Nobody begged him, nobody held a gun to his head.”[11] Volunteering is a choice; nobody owes you comfort; if it is too hard, quit. Three answers. First, unpaid work is still work: Terranova described it 25 years ago — labor “voluntarily given and unwaged, enjoyed and exploited”, on which the entire digital economy feeds[12]; the choice to enter does not make the extraction fair, and the industry that ships this code in its products is no innocent third party to the dispute. Second, the argument proves too much: if the worker’s being a volunteer sufficed to put his conditions beyond all criticism, no volunteer organization could ever be held abusive: conditions are a property of the system, not of consent. Finally, even those indifferent to the volunteer’s fate have an insurance interest: xz-utils was running on their machines too. The worker’s comfort is not a feel-good extra that projects will treat themselves to when everything goes well; it is a survival condition of the software, on the same footing as its tests.

But the fact is there, and the public records characterize it: the “community” is not a group of people who care for one another — what sociology has called a community since Tönnies (1887): bonds maintained for their own sake, where the person counts for more than his function[13] — but an interest group that gets

excited about the same things, wants its toy software bristling with exciting novelties on fixed dates, and does not care about the working conditions of those who produce it, as long as the facade holds. Volunteering is instrumentalized there by turns:

- to excuse the worker's incompetence and negligence — it's volunteer work, we do what we can;
- to wave away any responsibility of the worker for the consequences of his work, on users and colleagues alike;
- to wave away any responsibility of the community for the worker's working conditions;
- as a global, unreserved consent to any and all working conditions, as long as the worker stays. Otherwise he is invited to leave — but leaving the product of his work behind for everyone's enjoyment, in a form of Marxist alienation Marx had not foreseen[14].

The crudest of these uses have just been quoted, verbatim and archived; the full inventory counts 22 occurrences, on two platforms, over four years[10]. This is the endemic doublespeak of free software, the one the prologue announced: a libertarian, benevolent, emancipatory, collectivist facade — until you criticize it, confront it with its contradictions, or grow tired of endlessly mopping up, for want of structure and method, the incompetence of "colleagues" who do not improve. Then it shows its true nature: individualist, consumerist, exploitative and violent. And it sustains itself on the defeatist belief that it's like that everywhere and nothing can be done about it — when the degradation of a project like Darktable is measurable, dated, attributable:

1. to decisions made at a given moment and never re-examined since their effects became measurable — the chapter "Ship, don't dump" dated the decision, and the chapter "Non-heroic ambition" showed the founders' own diagnosis, left without follow-through;
2. to individuals who refuse to measure the impact of their contributions on technical debt and on the stability of the whole — the chapter "The cargo cult" did the arithmetic — and who entrench themselves behind the belief that doing better was not possible: a tautology if ever there was one;
3. to false beliefs about what sustains contributors' motivation — refuse nothing, plan nothing, ship two releases a year instead of one — when the projects that last do the opposite: the normal refusal rate of a healthy project is measured in the chapter "Non-heroic ambition", and what separates the free-software projects that live from those that die, measured on more than a hundred thousand projects, comes down to governance and stated goals — not to the nature of the task[15].

None of this is a property of the task at hand; all of it is a property of the beliefs of the people present, and of their way of working. None of it, therefore, is inevitable.

This chapter has made the worker's comfort a criterion; it remains to show that it can be upheld, concretely, in a real project. Everything above — the exposure, the finite availability, burnout as a property of the system — calls for a practical answer: an organization that makes the work reliable and, in

time, lighter. That is the subject of the next chapter: working so as not to have to work things over again.

---

1. Measured on the project's public Matrix room (log cited in the chapter "The closed circle"), counting addressed replies — quotes and mentions by name. In 2022, out of 4,905 addressed replies, the top 3 responders are an experienced user (30%), me (14.5%, 85 distinct interlocutors — on top of development), and the forum moderator (11%); the project's founder: 2.5%; the principal merger: zero addressed replies, 36 messages in 4 and a half years. In the following years, support is carried by power users and moderators — never by those who decide the design. A heuristic: unaddressed replies escape the count; the relative ranking, though, is robust. The aggregate of the two public venues gives the full arc of this load: on the community forum, my posts go from 44 (2017) to 191, 401, then 830 at the peak of 2020, 462 in 2021 and 233 over my last half-year — 2,161 in all (full count via the API, all categories: peer help, announcements and debates combined); on the chat room, 709 addressed replies over the 7 documented months of 2022. And the measure must be bounded in the other direction, because it checks out: peer help *in breadth* — being present in as many threads as possible — was always carried by others, the forum moderator first among them every year from 2017 to 2025; on that ranking, I appear in the top 5 only one single year (2020). My own form was depth and teaching: 37 threads opened, including the reference threads — the filmic FAQ

and its 222 posts, the tone equalizer thread — which the per-thread measure does not count, and which the chat room was still citing as an answer years after my departure. The French-speaking forum (Darktable.fr) documents the earlier period: 1,143 posts from 2016 to 2022, of which 493 in the single year 2018 — 4th responder of the forum that year, 6th in 2019 — before my activity migrated to the English-speaking channels; last visit recorded by the platform: July 24, 2022. On the expert domain — color science, filtered by keywords — a third of the chat room's messages were mine, 3 times more than the next contributor; and years after my departure, the chat room's stock answer to color questions was still a link to my videos. After September 2022: zero everywhere — and the chat room's total peer help, all people combined, falls by two-thirds the following year, never to recover (chapter "The closed circle"). ↩

2. Zhang *et al.* (2025). Longitudinal study of donations to maintainers (GitHub Sponsors). *Journal of Software: Evolution and Process* https://doi.org/10.1002/smr.70033 — a significant effect on short-term activity, no long-term effect; the sponsorship literature also notes the risk of crowding out intrinsic motivation. ↩

3. Eghbal, N. (2020). *Working in Public: The Making and Maintenance of Open Source Software*. San Francisco: Stripe Press https://press.stripe.com/working-in-public — the maintainer's attention as an overexploited common resource is the core of her analysis. See also, by the same author, *Roads and Bridges: The Unseen Labor Behind Our Digital*

Infrastructure. Ford Foundation, 2016. https://www.fordfoundation.org/learning/library/research-reports/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure/ ↩

4. Raman, N., Cao, M., Tsvetkov, Y., Kästner, C. & Vasilescu, B. (2020). "Stress and Burnout in Open Source: Toward Finding, Understanding, and Mitigating Unhealthy Interactions". *ICSE-NIER 2020*, ACM https://doi.org/10.1145/3377816.3381732 — the toxic interactions → burnout mechanism; the study provides no prevalence figures, and is not cited here for that. ↩

5. Graham, P. (2009). "Maker's Schedule, Manager's Schedule". Online essay (paulgraham.com) https://paulgraham.com/makersschedule.html — the maker's schedule in half-days, against the manager's schedule in hours: an ill-placed meeting costs half a day of making, not an hour. ↩

6. My contributions, 2019-2020-2021-2022, sources: git history (deduplicated aliases), GitHub API, the APIs of both forums, chat-room log. Commits: 197, 281, 210, 68; of which fixes: 68, 39, 38, 20; PRs opened: 83, 69, 52, 13; PRs reviewed: 25, 62, 49, 17; issue comments: 199, 164, 78, 39 — triage collapses as early as 2021 (−52%), a year before the code; Darktable.fr posts: 317, 191, 67, 25; pixls posts: 401, 830, 462, then 233 over the first half of 2022. Monthly 2022 commit profile: 18, 4, 0, 8, 32 (May, the month of the filter conflict), 5, 1, then nothing. Dated self-declarations, checked against the measures: "I have had like 2 weeks of vacation since last

christmas" (chat room, August 8, 2022); "I have time to review nothing" (same evening — 17 reviews that year, against 62 in 2020). A contribution measure, not a clinical diagnosis. ↩

7. Git history of the Darktable repository, commits excluding merges, deduplicated authors, as of July 22, 2026. Core departures: 15 commits or more in year n−1, zero commits in year n, 2016-2025 (18 to 22 cases depending on the alias-matching method — the profiles are established on the 22). Classification over the last 8 active quarters, by the ratio of the last two to the first two: 3 gradual declines, 16 dead stops (of which about 10 "flare-ups": near-empty history, one or two intense quarters, disappearance), 3 mixed profiles. The 4 profiles quoted, last 6 quarters (recomputed July 23, 2026, stable across alias matching): pillar 2010-2020, 2,127 commits, 4-10-18-27-0-2; meteor 2014-2016, 841 commits, 109-110-49-172-157-9; member 2018-2022, ≈ 550 commits, 33-22-27-101-15-3; doyen 2012-2025, ≈ 1,370 commits, 19-3-0-2-0-19. The flare-up quoted: 20 commits, all within a single quarter of 2023, with no activity before or after. Survivors' load: median commits of members at 15 commits and more — 55, 39, 35, 55, 58, 65, 35, 61, 74, 62, then 41 for 2016 → 2026 (partial). Methodological caveat: commits measure neither review nor triage, and a dead-stop profile can also stem from a life event; it is the rarity of gradual declines, not each individual case, that carries the conclusion. ↩ ↩

8. The 822 source files (C, C++, OpenCL) present at the head of the Darktable repository, full history excluding merges,

deduplicated authors: 7% have a single author, 13% at most two; 323 (39%) have a principal author who has committed nothing since 2025; the smallest group of principal authors covering half the files — the "bus factor" of the literature — counts 6 people. The counts weight the file's entire history: they therefore underestimate the recent loss. ↩

9. CVE-2024-3094 (March 2024): backdoor in xz-utils/liblzma, inserted by a contributor who had obtained co-maintainership after a years-long pressure campaign — multiple accounts urging the original maintainer, who declared himself exhausted and struggling, to hand over. Discovered by chance before general deployment. https://nvd.nist.gov/vuln/detail/CVE-2024-3094 ↩

10. Verbatim, discuss.pixls.us. Thread "r/Darktable options and path forward", 2022-06-29: "Yes, he had a lot of knowledge. But if he doesn't want to be here, nobody is forcing him to stay" (https://discuss.pixls.us/t/r-Darktable-options-and-path-forward-and-user-point-of-view/31388/57; archived: https://web.archive.org/web/20260721230849/https://discuss.pixls.us/t/r-Darktable-options-and-path-forward-and-user-point-of-view/31388/57); and thread "AP's new video on Ansel", 2026-06-20: "He talks about having too much free time, but who had almost six years to mostly write code?" (https://discuss.pixls.us/t/aps-new-video-on-ansel/58735/10; archived: https://web.archive.org/web/20260721230927/https://discuss.pixls.us/t/aps-new-video-on-ansel/58735/10). Both messages come from the same forum moderator. Full inventory, with

links and context: 22 occurrences of this rhetoric on two platforms, 14 distinct authors, 2022-2026 (survey of July 22, 2026). ↩ ↩

11. Verbatim, r/DarkTable (Reddit), thread "Ansel: Color controls finally correct", April 2026: "Who pushed him? He pushed himself. It was all volunteer work, it was all things he wanted to do. Nobody begged him, nobody held a gun to his head." (https://reddit.com/r/DarkTable/comments/1sdwjxo/comment/oeofv15/ ; archived: https://web.archive.org/web/20260721231145/https://old.reddit.com/r/DarkTable/comments/1sdwjxo/ansel_color_controls_finally_correct/oeofv15/). In the same thread, a third party pointed out the contradiction: "When it comes to working for free on software, it's his own choice and everything is his individual responsibility, so he should shut up and nobody owes him anything. But when it comes to providing services to users, on websites he pays for, suddenly he has responsibilities ? Choose one, but right now you are contradicting yourself." (https://reddit.com/r/DarkTable/comments/1sdwjxo/comment/oep7213/ ; archived: https://web.archive.org/web/20260721231207/https://old.reddit.com/r/DarkTable/comments/1sdwjxo/ansel_color_controls_finally_correct/oep7213/). ↩

12. Terranova, T. (2000). "Free Labor: Producing Culture for the Digital Economy". *Social Text*, 18(2), 33-58

https://doi.org/10.1215/01642472-18-2_63-33 — free labor, "voluntarily given and unwaged, enjoyed and exploited". ↩

13. Tönnies, F. (1887). *Gemeinschaft und Gesellschaft*; Fr. trans.: *Communauté et société. Catégories fondamentales de la sociologie pure*, Paris: PUF, 2010. Community (*Gemeinschaft*) holds through organic bonds maintained for their own sake — kinship, neighborhood, mutual support — whereas society (*Gesellschaft*) aggregates individual interests that coexist without mutual obligation. The vocabulary of free software says "community"; its records describe a society. ↩

14. Marx, K. (1844). *Economic and Philosophic Manuscripts of 1844*, first manuscript, "Estranged Labour": the product of labor becomes for the worker an alien object, a power independent of him. Free-software volunteering adds the case Marx had not envisaged: an alienation without wage and without coercion, consented to — the labor "voluntarily given and unwaged, enjoyed and exploited" of Terranova (note above), pushed to its conclusion: the worker may leave, the product stays. ↩

15. Schweik, C. M. & English, R. C. (2012). *Internet Success: A Study of Open-Source Software Commons*. Cambridge, MA: MIT Press https://mitpress.mit.edu/9780262017251/internet-success/ — an analysis of more than 100,000 projects hosted on SourceForge; the factors that separate the projects that succeed from those that are abandoned pertain to how the project is run — leadership, vision, stated goals, observed usefulness — and not to the domain or the difficulty of the

task. Already cited in the chapter “The committee will invent nothing”. ↩

22. Working for the work

The previous chapter laid down a principle: work must capitalize, or it devours the worker. Capitalizing here means that work must produce knowledge about itself, methods and tools that will make future work easier and faster by drawing on past experience. This chapter shows how, and it is the most concrete in the book: an experience report more than an argument.

But producing knowledge about your own work does not come by itself: it requires working with a method — something that can be validated, falsified, and amended. Work that nothing of the sort frames produces no knowledge about itself: every new task started without a preconceived idea of the sequence of subtasks to complete and validate along the way leaves the project as ignorant as it was before, because without a criterion set in advance, there is no telling what worked, or why, or what the next attempt should change or keep. And this is not a figure of speech, it is a theorem: in optimization, the so-called "no free lunch" results establish that on average, over the set of all possible problems, every search strategy — random draws included — performs exactly the same; you only beat chance by building into your strategy true knowledge about the structure of the problem[1]. Method is nothing but that knowledge, accumulated, verified, and transmissible — Simon already described design as a search through a space of possibilities, made tractable by the heuristics that prune it[2]; and the

falsifiability on which that knowledge rests was laid down back in the chapter "The word and the thing". A collective that works without method therefore explores its solution space the way a neural network in training does: by blind trials, in bulk. Except it does so at human speed. The machine can afford brute force because it tries millions of times per second — the chapter "Intelligence upstream" showed what you get out of it when it is harnessed wisely; at the scale of a team of evening volunteers, the same strategy goes by another name: eternity. Dawkins made the arithmetic demonstration that has remained a classic: a 28-character sentence typed at random has about one chance in 10^{40} of coming out — at that rate, billions of billions of times the age of the universe are not enough; the same sentence, searched for by variation *with retention of what works*, falls out within a few dozen generations[3]. Retention of what works: that is the minimal definition of method, and everything that follows — the triage of requests, the board, the automation, the teaching — is nothing but that retention, organized. Without it, a project capitalizes nothing: it replays the same draw over and over, and calls that experience.

But you still have to know what a working method even is, and I learned it during my third-year project in engineering school. I was working for a start-up that built thermal control for wine cellars of every size, from a simple fridge to a full walk-in cabinet. A fridge does not have a uniform temperature inside its enclosure — the cooling element is somewhere, not everywhere — and the problem gets worse with size. The project was to evaluate, then design, numerical simulation methods able to

reconstruct the full temperature map inside the enclosure, from thermodynamic models and a few probes — in real time, on a microcontroller with limited computing power. So the trade-off between accuracy and computational cost had to be steered, something an ordinary simulation on a compute workstation never has to do. The professor supervising me had no more of an answer than I did: the problem was singular enough. It was watching him unroll the method in front of me, cold and unprepared, that made me understand the difference between him and me. Neither of us had the solution, but he knew which end to attack it from. "Start by trying A. If it works, refine with B. If it doesn't work, look at C. On Google Scholar, the keyword for your problem is X — look at Y too." Within a quarter of an hour, I had the flowchart of solutions to test, in what order, where to look for them, and a fallback position for every attempt. I still had no solution; I had a battle plan to get there. That is what a method is. Whether solution A worked or not, I would in either case learn something about it and something about my problem, and I already knew what to do at the next step: every attempt became an experiment in the proper sense — falsifiable, instructive in both of its outcomes. Without a method, I would have laboriously run searches with the wrong keywords, cobbled together some code, and badly reinvented reduced models the literature had already vetted. And even the complete failure of the whole tree would have said something precise: that the problem was ill-posed from the start, in the wrong terms or with the wrong criteria — the chapter "Stating the problem" showed what that diagnosis is worth. Solution or no solution, whatever happened, I would end up with a better understanding of the

problem. The professor's knowledge was not the answer: it was the pruning of the solution space — a quarter of an hour of problem structure instead of blind random trials.

That lesson comes from a research project, but it transfers to any work, as soon as you accept to make the work itself a research subject: seeking the best way to accomplish the task, through trials that teach something in every case, so as to work better, and less, the next time. Yet for many people, experience amounts to repeating the same gestures for decades without variation and without questioning them, which is the mark of experience devoid of experimentation — in other words: a habit you cannot explain.

For years, I carried my software more or less alone — development, releases, support, documentation. At that scale, you do not hold on by working more: you hold on by working *for the work*, that is, by devoting a share of the hours not to the day's task but to the tooling, methods, and organization that will make tomorrow's task more reliable, then unnecessary. Each hour recounted here did not solve a problem: it removed a category of problems. It is the exact opposite of the regime described in the chapter "The invisible worker", where the further you go, the more you break.

Everything starts at the door, because the door is where everything overflows: requests. My project's rule is brutal, and it follows from the chapter "Listening without obeying": *feature requests are not taken as they come.* It is the designer who consults users about their needs, when a design workstream

opens — not the users who petition the designer at the whim of their wishes. There are two reasons for this. The first is cognitive: a request that lands in the middle of a workstream is an interruption, and we will see below what an interruption actually costs. The second is a matter of method, and the chapter "Listening without obeying" established it: a request phrased as a solution — "add this button" — is inadmissible; only the request phrased as a need counts, situated in the workflow of a trade — "I must be able to accomplish this task." Filtering at the door is not contempt for the user: it is refusing to build Frankenstein's creature one graft at a time. Requests that fit no open workstream wait until one opens; those that are only solutions in disguise are translated back into needs, or refused.

Then comes the organization of the work itself, and it fits in a few rules, all of them written down. Work in progress is displayed on a kanban board: to do, in progress, to test. The tool comes from industry — at Toyota, kanban was born as a system of visual signals serving precisely to limit the work underway and to pull production from downstream rather than push it from upstream[4] — and its transposition keeps the essential: making work in progress visible, hence limitable. In my case, the limit is two programming workstreams open at the same time, never more, chosen *independent* of one another: you switch to the second while the first waits for usage feedback, which keeps the mind busy without scattering it; and since the two workstreams do not touch, you always know which one introduced which regression. It is the opposite of the hive of the chapter "The committee will invent nothing": few fronts, held one after the

other. Finally, in the middle of a workstream, I do not handle subjects foreign to it, and I do not listen to them either. This is not a whim: work psychology has measured what interruption costs — interrupted work gets caught up by working faster, but at the price of markedly higher stress, pressure, and effort, and merely resuming a complex task that was dropped takes long minutes of re-immersion[5]. Programming is work that exists only in blocks: a broken block is not half done, it has to be redone. This discipline looks rigid; it is what allows one man alone to carry a team's load.

The next tier is automation, and it is the heart of "working for the work": every chore repeated by hand is a debt that can be paid off once and for all. The installation packages for Windows and Linux build themselves, every night, on the sole condition that code was pushed the day before, and their links get published to a dedicated channel: any user can try, that very evening, what was written during the day, without anyone having compiled or published anything by hand. The idea is not new — the daily build with smoke test is one of the oldest documented good practices of software engineering, and continuous integration is its heir[6] — but the chapter "Ship, don't dump" made it a testing requirement; here, it is a machine that runs without me, and that makes all the difference. Same logic for the denoiser of the chapter "Finishing and maintaining": it was not shipped as a frozen neural network, but as a chain — image sorting on the users' side, validation on the maintainer's side, automated retraining — so that improving the model next year will not require this year's work all over again. Every time, the question is

not “how do I do this task?”, but “what must be built so that this task, next time, gets done without me?”.

The extreme case of this logic goes after the most insidious source of exhaustion from the chapter “The invisible worker”: repetitive communication. Having become the public face of the project, I was spending an unreasonable amount of time answering questions whose answer already existed, written by me, somewhere on one of my sites: the documentation was getting lost in the limbo of the Internet, and the maintainer served as a search engine made of flesh and bone. On the forums, well-meaning volunteers wear themselves out at the same task: repeating the same information, pasting the same links back in, for endless waves of beginners. Yet these beginners are not lazy; they are running into a problem that ergonomics measured 40 years ago under the name of the *vocabulary problem*: the probability that two people will spontaneously call the same thing by the same word is below 20%[7]. A classic search engine requires knowing the keywords; the beginner, by definition, does not know the jargon that would give him his entry point into the information system. So I built the missing tool. First a virtual secretary that sorts the incoming mail; then, on top of it, a semantic search engine — I named it Chantal — that learns the kinships of meaning between words, retrieves an answer through meaning rather than through the exact word (symbol), and above all *suggests* to users the keywords neighboring their question: it hands them back the entrance to the maze. Trained on close to a million pages drawn from my sites, from the documentation, and from reliable sources, indexing 400,000 pages in two languages, it

answers before anyone needs to disturb me. The technical lesson is worth noting, because it sums up the book: 80% of the effort went not into the learning model but into cleaning the data — garbage in, garbage out — and a model tuned too finely to its corpus makes a worse search engine, not a better one. The bulk of the work was, once again, invisible and thankless; it pays itself back with every question I no longer have to answer.

And the tool followed the natural slope of this logic, which is the real lesson of the chapter: it stopped being mine. Designed at first to relieve support — finding again the documentation pages that the same questions kept asking for — the engine then swallowed the development documentation, then the resources of the trade itself, the OpenMP and OpenCL specifications among others, then the entirety of the repository's commits, pull requests, and issues — then the academic literature of the field: sensors, color science, sociology of the image. What had been my firewall became everyone's knowledge infrastructure — no longer a software index, but a specialty portal on imaging, anchored in a software project: users find their manual there, today's contributor his specifications and interfaces, tomorrow's contributor the history of the decisions that preceded him, and the curious the science that grounds it all. So "working for the work" must be stated in its full extension: you do not work only for your future self — you work for your present and future colleagues, for whoever will one day have to keep the machine running. A well-built workflow does not only serve its author: it lowers the cost of work for everyone who shares it, including those who have not arrived yet. And I will take the idea to its

end: this is the deepest form of respect for people, users and contributors alike. Not the respect of pleasantries, but respect in deeds: making everyone's work easier, and making everyone able to do it without you. The autonomy of others is the only homage that costs nothing to those who receive it and that grows with time; the conclusion of this book will say that this kind of respect goes by another name.

There remains the most misunderstood source of load, and its treatment belongs to the chapter "There is no technical problem": a large share of what arrives disguised as a bug report or a feature request is in reality a training deficit. A poorly trained user has false expectations; taking his requests literally means loading the software with functions that duplicate what already exists and weigh the machine down for nothing. That problem does not get fixed in code: it gets fixed at the root, by teaching. Hence an investment few projects consent to: documentation that follows the order in which functions appear in the interface, video tutorials, guides organized by trade task rather than by module, and theoretical resources so that users learn to diagnose on their own. Teaching is still "working for the work": every page that teaches a user to troubleshoot alone dries up a source of solicitations. The peace of the worker of the chapter "The invisible worker" is not obtained only by protecting yourself; it is obtained by making the user autonomous — the same gesture, seen from the other end.

The objection is predictable, and I have heard it: all this time spent tooling up, automating, documenting, filtering is time not spent writing the software — meta-work, a luxury that

overwhelmed projects cannot afford. The answer is arithmetic, and it makes tangible the wager of the chapter "The invisible worker": tooling is an investment with compound interest. The hour invested in the nightly build pays itself back every night; the hour invested in Chantal, with every question avoided; the hour invested in triaging requests, with every Frankenstein's creature not built. Refusing to tool up means choosing to pay full price forever for every repeated task — and that is exactly how a maintainer burns out: not through an excess of real work, but through the repetition of work he could have eliminated. To "we'll see when we have time," this chapter answers: you never have the time *because* you do not tool up. The loop only breaks by working, for a while, for the work rather than for the task.

The yield of this organization can be measured — at a scale not comparable to that of the chapters "Ship, don't dump" and "The committee will invent nothing": Ansel receives 10 times fewer reports than Darktable. But it is precisely the regime that compares, not the volume. Of the 717 issues received since 2022, 92% are closed — against barely more than half in Darktable's recent cohorts — and not a single one by a bot: the project has no stale bot; every closure is a human act. Triage exists: regression, priority on 4 levels, crash backtrace attached, "unclear" — the triage taxonomy covers most of the record, where 44% of Darktable's recent issues carry no label at all. The median time to close is slower — 7 days against a day and a half — and that is the price of the regime: Darktable closes fast what it closes and abandons the rest to the bot; here almost everything gets closed, hence sometimes late — a tenth of the cases wait more than a

year. And over 2026, a peak year with 300 issues received, more were closed than came in: the backlog fell to two. This is no virtue: it is the arithmetic of a filtered entrance, of triage done on arrival, and of a refusal of the bot. The same flow of issues, organized, ceases to be a torrent you endure[8].

But it must be said why this loop almost never breaks in free software, because the reason is structural, and it carries further than tooling. The volunteer contributor comes in the evening, after his day's work, and he comes for one precise thing: the joy of building — the chapter "Finishing and maintaining" described that euphoria, and the chapter "What insularity produces" will put figures on what the surveys confirm: the first declared motive for contributing is intellectual stimulation, play. Now, triaging requests, writing rules, maintaining a build chain, cleaning data, documenting: that is precisely what his day job looks like — process, administration, thankless tasks. He fled it. Asking a volunteer to do in the evening, for free, what he is paid to endure during the day is asking water to flow uphill. And I am not putting words in their mouths: as early as 2016, on the community forum, to a user asking that the existing software be stabilized rather than projects piled up, a contributor replied that "the vast majority of us are not getting paid to hack on open source software" and that it is "inappropriate to tell them to work on something more useful" — leisure claimed as an exemption clause, in black and white[9]. Nobody does it, and it is not laziness: it is the very construction of the model. A system that recruits on pleasure gets exactly what pleasure funds — new features, exciting prototypes — and nothing of what makes a tool

robust, because robustness is precisely the part of the work that is no fun. This is my hardest diagnosis, and this chapter makes it inescapable: the contribution model of free software, founded on volunteering, is structurally doomed — not for lack of talent or goodwill, but because it selects, by construction, the pleasant work, and orphans the work that matters.

The model fails, for that matter, even on its own ground. Darktable stated its rule at a conference, word for word: "if there is a rule, it's don't stop anyone from coding when they're having fun"[10] — the chapter "The closed circle" will come back to this doctrine. It promised, through pleasure, a perpetual influx of hands. But the series of the chapter "The invisible worker" say the opposite: commits, pull requests, and reviews down by two thirds, the working core shrunk from 27 to 13 people between 2020 and 2025. My theory is that the code itself became the repellent: years of features merged without architecture — the chapter "The cargo cult" did the arithmetic of those merges without review — make a plate of spaghetti nobody feels like untangling. I verified it without meaning to: I set out to recruit two seasoned developers I knew; they opened the code, and they said no thanks. Programming stops being a pleasure when the program is unreadable. So the single rule beat itself: by dint of protecting everyone's immediate pleasure against any discipline, it manufactured a code base nobody wants anymore, and recruitment dries up exactly where everything had been organized for it. You do not get out of this by exhortation; you only get out through a structure that pays for, or mandates, the thankless work. That is what the last part will build.

This chapter thus closes on a note that tempers its own diagnosis, and that is deliberate: the free-software collective often fails, but it is not condemned to. The committee invents nothing — a lead with tools does; the invisible worker gets crushed — a project that capitalizes protects him. But all these rules — the triage, the board, the automation, the teaching — presuppose someone with the mandate to write them and make them hold. We land back on the question this book keeps crossing: by what right, in what structure, with what legitimacy? The previous parts have shown what it costs not to answer it. Before getting there, one question remains that this chapter has left untouched: of all this work we have just organized, which of it should have been accepted in the first place? That is the subject of the next chapter.

---

1. Wolpert, D. H. & Macready, W. G. (1997). "No Free Lunch Theorems for Optimization". *IEEE Transactions on Evolutionary Computation*, 1(1), 67-82 https://doi.org/10.1109/4235.585893 — "the average performance of any pair of algorithms across all possible problems is identical", including when one of the two is the random draw; and for one strategy to beat another, the structure of the problem "must be known and reflected directly in the choice of algorithm". ↩

2. Simon, H. A. (1996). *The Sciences of the Artificial*, 3rd ed. Cambridge, MA: MIT Press, ch. 5 ("The Science of Design: Creating the Artificial") https://mitpress.mit.edu/9780262691918/the-sciences-of-the-artificial/ : design as selective search through a space of

possibilities, the allocation of search resources, and "satisficing" as the stopping criterion — one seeks a sufficient solution, not an optimal one, because the space exceeds any exhaustive exploration. ↩

3. Dawkins, R. (1986). *The Blind Watchmaker*. Harlow: Longman, ch. 3 ("Accumulating Small Change"); French trans. *L'Horloger aveugle*, trans. B. Sigaud, Paris: Robert Laffont, 1989. The target sentence METHINKS IT IS LIKE A WEASEL — 28 characters over an alphabet of 27 signs — has one chance in $27^{28} \approx 10^{40}$ of being typed in one go; Dawkins puts single-step selection at around 10^{30} years, more than a billion billion times the age of the universe. *Cumulative* selection — random variation, but retention of the best candidate at each generation — reaches the same target in 43, 64, and 41 generations over his three runs. ↩

4. Ohno, T. (1988). *Toyota Production System: Beyond Large-Scale Production*. Cambridge, MA: Productivity Press, ch. 2 https://doi.org/10.4324/9780429273018 (already cited in the chapter "The root and the symptom"; French trans. *L'Esprit Toyota*, Paris: Masson, 1989) — kanban, "an idea I got from American supermarkets" in Ohno's own words, is the steering tool of the system: its first rule is that the downstream process comes to withdraw from the upstream process, and its function is to prevent overproduction — the number of cards in circulation caps the work in progress, and control is visual, a "management by sight". ↩

5. Mark, G., Gudith, D. & Klocke, U. (2008). "The Cost of Interrupted Work: More Speed and Stress". *Proceedings of*

CHI 2008, ACM, 107-110 https://doi.org/10.1145/1357054.1357072 — interrupted work is made up for by speeding up, at the price of significantly higher stress, frustration, and effort. On resumption: Mark, G., González, V. M. & Harris, J. (2005). "No Task Left Behind? Examining the Nature of Fragmented Work". *Proceedings of CHI 2005*, ACM, 321-330 https://doi.org/10.1145/1054972.1055017 — when an interrupted task is resumed the same day, it is on average resumed only after 25 minutes, after passing through two other subjects (the "23 minutes" circulating in the press appears in no published article). On immersion: DeMarco, T. & Lister, T. (1987). *Peopleware: Productive Projects and Teams*. New York: Dorset House, ch. 10 https://openlibrary.org/books/OL2392791M/Peopleware (French trans. of the 1st ed.: *Les Hommes de l'ordinateur*, Paris: Masson, 1991) — entering flow "takes a slow descent into the subject, requiring fifteen minutes or more of concentration before the state is locked in", hence their E-factor: uninterrupted hours, not hours of presence, measure the real work. ↩

6. McConnell, S. (1996). "Daily Build and Smoke Test". *IEEE Software*, 13(4), 143-144 https://doi.org/10.1109/MS.1996.10017 — "The daily build and smoke test process keeps integration errors small and manageable, and it prevents runaway integration problems"; extended by Fowler, M. "Continuous Integration". martinfowler.com, 2000, rewritten in 2006

https://martinfowler.com/articles/continuousIntegration.html. ↩

7. Furnas, G. W., Landauer, T. K., Gomez, L. M. & Dumais, S. T. (1987). "The Vocabulary Problem in Human-System Communication". *Communications of the ACM*, 30(11), 964-971 https://doi.org/10.1145/32206.32212 — "In every case two people favored the same term with probability <0.20"; and "the popular approach in which access is via one designer's favorite single word will result in 80-90 percent failure rates in many common situations". Their remedy, "unlimited aliasing", is the ancestor of the search by meaning described here. ↩

8. aurelienpierreeng/ansel https://github.com/aurelienpierreeng/ansel, as of July 23, 2026: 717 issues created since 2022 (36, 159, 54, 160, then 308 per year), 658 closed (92%), 59 open; in 2026: 308 created, 339 closed. Median time to close 7.4 days, 75th percentile 99 days, 90th percentile 399 days. No automatic stale label exists in the tracker; main labels: *regression* (83), priorities (critical 20, high 78, medium 63, low 54), *backtrace: included* (48), *unclear* (33). 138 distinct reporters, 32 of them with 3 issues or more; 8% of the issues are filed by the maintainer himself (the kanban board described above runs through the tracker). 19% of the issues have no comment at all — a share mostly closed by the fix itself: the records of a two-man project do not converse for conversation's sake. ↩

9. https://discuss.pixls.us/t/975, March 2016 (archived; the thread is analyzed in the chapter "What insularity

produces”): “the vast majority of us are not getting paid to hack on open source software. As long as people are doing all of this in their spare time I think it’s inappropriate to tell them to work on something more useful”. ↩

10. Word for word: “if there is a rule, it’s don’t stop anyone from coding when they’re having fun” — the “do-ocracy” slide of the project’s official presentation at the Libre Graphics Meeting 2019 (video, at 17 min 31 s https://www.youtube.com/watch?v=o4cnoNMdZBw), whose three key slides the chapter “The closed circle” analyzes — the rule, the responsibility, and the balance sheet that ends on “technical debt :/”. ↩

23. Non-heroic ambition

There are two ways to be ambitious for a piece of software. The first is visible: a spectacular feature, announced, applauded on release day. The second is not: the software still works 10 years later, and nobody ever had to notice. The first is heroic. The second is an ordinary heroism, silently useful, discreetly effective.

The chapter "Finishing and maintaining" laid down the rule: undertake nothing you lack the means to finish and maintain. It left two questions open, and they are the ones that matter in practice. At the moment of accepting a feature, how do you know what it will cost to keep? And who has the power to say no? This chapter answers with the Darktable record in hand, so that the answer may serve those who refuse to end up the same way.

First, maintenance must be stripped of its status as a punishment. We speak of it as the price of badly written code: properly designed, software supposedly would not need any. That is false, and the question was settled half a century ago. Between 1974 and 1996, Meir Lehman formulated 8 empirical laws of software evolution; the 3 best-validated say that a program in real use must change continually, that its complexity increases, and that its perceived quality declines[1]. The clause attached to the third matters more than the law itself: quality declines *unless specific work is invested to prevent it.*

In other words, degradation is the default state. Software you stop maintaining does not stay where it is: it comes apart in place, because everything moves around it — the operating system, the libraries it depends on, the file formats, the machines, the usages. Even in the ideal case, maintenance is not the repayment of a past fault: it is the rent of existence. And rent is budgeted in advance. Obviously, if the price of past faults is added on top, the budget explodes.

It remains to know how much. The answer is known, stable, and regularly ignored: 60 to 80% of a software product's total lifecycle cost comes after delivery. The founding study by Lientz and Swanson, conducted across nearly 500 organizations, gives the breakdown: so-called *perfective* maintenance — adapting to needs that have changed — accounts on its own for more than half the effort, and three quarters of maintenance work consists not of fixing failures but of keeping up with the movement of the world[2]. These proportions have held for 40 years, through every change of technology.

The consequence is brutal for the way decisions get made. When a team discusses a feature, it discusses what the feature will cost to write. Yet that cost represents a quarter or a third of the real outlay. The rest will come later, will be paid by someone else, and will never have appeared in the estimate. Deciding on a feature by its development cost is debating its price while knowing only a quarter of it.

If maintenance is to this extent the law of the environment and the bulk of the expense, it remains to explain why it is despised.

The answer is not technical: it lies in what this work *shows*. It is repetitive. It produces nothing new. It does not announce itself — nobody publishes "this week, the software kept working." It consists of going back over someone else's work, understanding before touching, restoring what already existed. In the ordinary division of labor, these properties designate a precise category of tasks: the ones you delegate. To women, for what belongs to the household; to subordinates, to so-called support staff, to the less qualified, for what belongs to the firm. The chapter "What insularity produces" quoted the sharpest formulation of it: the obsession with technological novelty obscures all the labor — housework included — that women disproportionately perform.

That displacement is worth remembering, for it reaches beyond maintenance: work universally required, historically assigned to subordinates, and devalued for that very reason — this is, word for word, the definition the ethics of care gives of care.

Yet this ranking is wrong, and wrong in a way that can be established. The chapter "Code is a liability" gave Kernighan and Plauger's rule: "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as you can, you are, by definition, not smart enough to debug it." The work being relegated is the harder of the two. The division is therefore not a hierarchy of difficulty: it is a hierarchy of visibility, and it ranks things backwards. The best programmers should be assigned to maintenance. But being among the best opens more prestigious possibilities.

The mechanism of this invisibility has been described elsewhere, and it is cruel. Susan Leigh Star showed that an infrastructure gets noticed only by breaking down, and that the work holding it up belongs to a category organizations know how to produce but not how to count: invisible work — invisible not by accident, but because the systems of description themselves make some tasks countable and others not[3]. Successful maintenance erases itself. Done well, it leaves nothing to show; having left nothing to show, it cannot be credited to whoever did it. The maintainer has only one way to prove his work: to stop doing it. You do not see a solved bug until it becomes one.

This is reminiscent of the superhero topos of popular culture — saving the world every week, alone, in one shattering feat. You never see the people who clear the rubble, reopen the hospital, restore the water and put the power lines back up — the work that nonetheless fills the months after every catastrophe, and without which the saved city remains uninhabitable. The genre's implicit moral never varies: what deserves telling is the one-off exploit of an exceptional individual, never the continuous upkeep by an ordinary collective. The comic-book superhero has his counterpart in professional mythology.

It remains to ask whether this distribution of prestige is an accident. I do not believe it is, and here is what I rest that on. The composition of the milieu is measured, and the chapter "What insularity produces" gives it: men, around fifty, trained as engineers, and 2% women. What these people do here, they do as volunteers, in the evening, with no one assigning it to them: their choices therefore reveal not constraints but dispositions. This is

what Bourdieu calls habitus — schemes acquired elsewhere, in a trade and a social position, and replayed here without ever being decided. It is easy to imagine that in a milieu whose members mostly occupy, in their paid jobs, the design side of the work — the side where upkeep is done by others — the division reproduced on volunteer time is the one they already know. Designing and announcing on one side; maintaining on the other, that is to say, in the shadows.

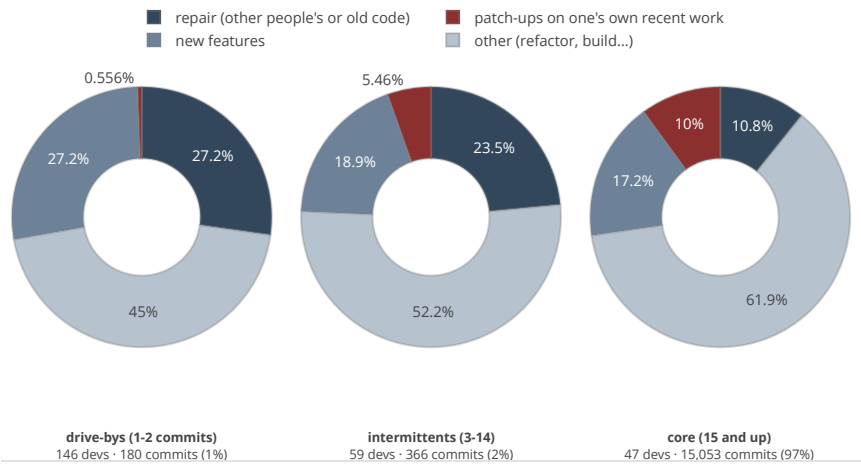
This division is not a sociological deduction: the repository puts numbers on it, commit by commit[4]. Since 2019, 252 people have modified the code. In the order of the questions — who repairs, how much, since when:

- **Who has never repaired**: 48% of developers have signed no fix since 2019; adding those whose fixes only patched up their own work of the preceding weeks, half have never repaired anything they had not broken themselves.
- **Participation follows commitment**: among the drive-bys at 1 or 2 commits, 32% repaired at least once; among the intermittents (3 to 14 commits), 56%; in the core (15 commits and up), 96%.
- **Volume is the core's business**: 47 people sign 97% of the commits and 92% of the true repair. The 146 drive-bys weigh 1% of the flow — their rare contribution is, as it happens, often a fix, but it weighs nothing in mass.
- **Half the "fixes" are not fixes**: of 3,289 commits declared as fixes, 1,531 patch up their author's own recent work — the tax on new features, not maintenance. This patch-up is 99% a

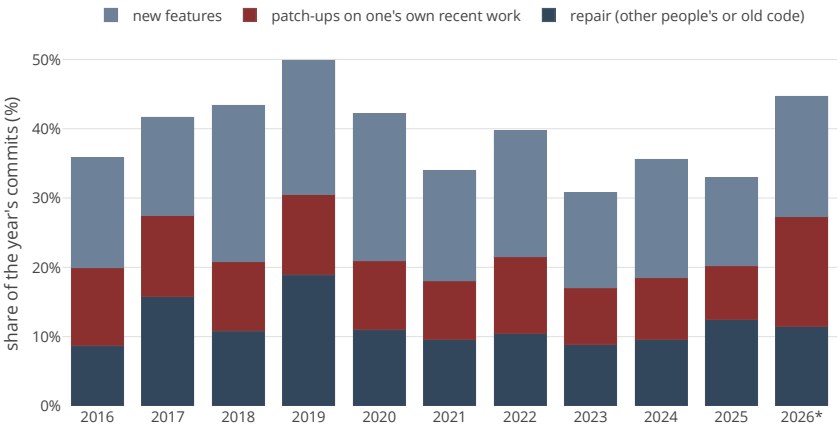
core phenomenon, mechanically so: you need recent work to patch up.

- **True repair — other people's code, old code — weighs 1 commit in 9** (1,758 out of 15,599), and this weight has been stable since 2016: between 9 and 19% depending on the year, with no trend. No era of the project, even at 83 active developers, gave more to upkeep.
- **What collapses is the volume, not the proportion**: about 350 commits of true repair per year at the 2019-2020 peak, 128 in 2025 (−60%) — at exactly the pace at which the core melted away.

And this last figure must be set against the literature this chapter opened with: 60 to 80% of a software product's lifecycle cost comes after delivery. Darktable has been in service for seventeen years — it sits entirely in that phase, the one where, by the industry's account, upkeep dominates the budget. It devotes one commit in nine to it, 11%, and keeps shipping new features every year. Lientz and Swanson's "maintenance" covers more than correction, admittedly — adaptation, perfection — and a commit is not a euro; but the gap is not a nuance of definition, it is an order of magnitude. A software product whose entire remaining cost is upkeep, and which gives repair a ninth of its energy while new features keep coming in, is not saving money: it is accumulating the debt this chapter has costed, and leaving it to others.



Who repairs? Composition of code commits by commitment tier, 2019-2026: repair of other people's or old code, patch-ups on one's own recent work, new features, and the remainder (refactoring, build, miscellany). The tiers do not carry the same weight: the drive-bys account for 1% of commits, the core for 97% — and it signs 92% of the true repair. The patch-up is a core phenomenon: it presupposes recent work to patch up.



The mix does not change: share of the year's commits classified as true repair, patch-up on one's own recent work, and new features, 2016-2026 (the remainder — refactoring, cleanup, build — completes to 100%; 2026 cut off at July 31). The proportions are stable across ten years: true repair oscillates between 9 and 19% with no trend, whatever the headcount. What this chart does not show is volume: about 350 commits of true repair per year at the 2019-2020 peak, 128 in 2025 — the proportion holds, the hands disappear.

So much for the contempt. The record now offers a textbook case. Between 2020 and 2022, the project I left undertook several ambitious overhauls: the keyboard-shortcut system, rewritten to make every action drivable from an external controller; the organization of processing modules into groups, turned into a configurable layout tool; the collection filters, that is, the commands that sort the photo library, overhauled twice in 4 months. This book has already given their line counts in the chapters "Beneath the interface, dragons" and "The committee

will invent nothing". These 3 workstreams share two properties, and they are the subject of this chapter.

The first: none of them answered a previously established problem. No specification, no measured pain, no identified audience — the chapter "Stating the problem" said what goes missing when all of that is missing. The second, graver: the ambition stopped at delivery. Nobody had costed what these overhauls would take to maintain, because nobody had to.

The result can be measured today, and it takes the form of a two-speed regime. Among the bug reports that do eventually get fixed, half close in 1.3 days; but the ninth out of ten, ranked by delay, already waits more than 22 days, and 47% of all reports will never be resolved at all, just closed by the stale bot[5]. This is not the distribution of an overloaded team processing requests in order of arrival. It is the signature of two distinct populations: on one side the easy fixes, settled within the day; on the other those that would require touching the architecture, that exceed the time and skills available, and that never get settled. Between the two, almost nothing. A healthy project has a queue; this one has a triage.

I supplied the testimony as it happened, in early 2023: "2022 is the first year that I find myself practically unable to identify the cause of most interface bugs, because the working logic has become very obfuscated and the code incomprehensible"[6]. An architecture you can no longer read is an architecture you can no longer repair. The ambitious features of 2020-2022 had not been enclosed in separate modules: they had been woven through the

rest of the program. Fixing one of them requires touching everything it runs through, and every fix adds one more interaction. The system does not get repaired: it gets more complicated by being repaired.

That would leave the honest exit, which would consist of withdrawing what you cannot keep. It is barred by the mechanism the chapter "In praise of the simple" described about options: what cost a lot to build looks too precious to throw away, even though the expense is made and will not come back. So you keep a half-broken feature that could nonetheless be made to work, and you pay the upkeep of the whole for the one half that works, denying the other.

So much for the finding. The hard question sits upstream: at the moment of saying yes or no, how do you assess what a feature will cost over 10 years? A good part of that assessment belongs to the experienced developer's intuition, and it cannot be demonstrated in a meeting. This is exactly Polanyi's *tacit* knowledge, invoked in the prologue: we know more than we can tell, and the engineer who senses that a workstream will go wrong cannot always produce the rule that tells him so. That intuition is acquired through long exposure to the consequences of one's own decisions — that is, by staying. Hence a consequence the chapter "The incompetent worker" will develop from another angle: the passing contributor, the one who stays a year or two and then disappears, cannot have it. Not that he is less gifted: he will leave before seeing the consequences of what he wrote come back down, and so he offers, in good faith, work whose future cost is structurally invisible to him. His departure

carries the knowledge away with him, besides — out of 1,932 free-software projects studied, 16% saw all of their key developers leave, and development was taken over by others in only 41% of those cases[7].

Conceding the share of intuition does not mean abdicating, because part of this assessment is standardized and another part can be measured. Standardized first: the ISO/IEC 25010 standard does not treat maintainability as a vague virtue; it breaks it down into assessable properties, the first of which is modularity — the degree to which one component can be modified without affecting the others. Then come analyzability, that is, the capacity to diagnose a failure and to assess the impact of a change before making it, then modifiability and testability[8]. These words supply a grid of questions to ask before accepting: what will have to be touched to change this? will we be able to tell where a failure comes from? will we be able to test it? These are not questions of taste.

Measured next, and this is the point believed to be out of reach. The "surface of interaction with the rest of the application" is not an image: it is a quantity. MacCormack, Baldwin and Rusnak formalized it under the name of propagation cost — the average proportion of the system's elements affected when you modify one picked at random. It is computed on a design structure matrix, that is, a table crossing all the components of the program to mark which depend on which. They applied it to real software, including Linux and Mozilla: Linux turns out markedly more modular, and Mozilla's great overhaul made its propagation cost drop, which gives the measure its value as a control[9]. A

project can therefore track its own modularity over time the way it tracks its test coverage, and see degradation coming instead of discovering it.

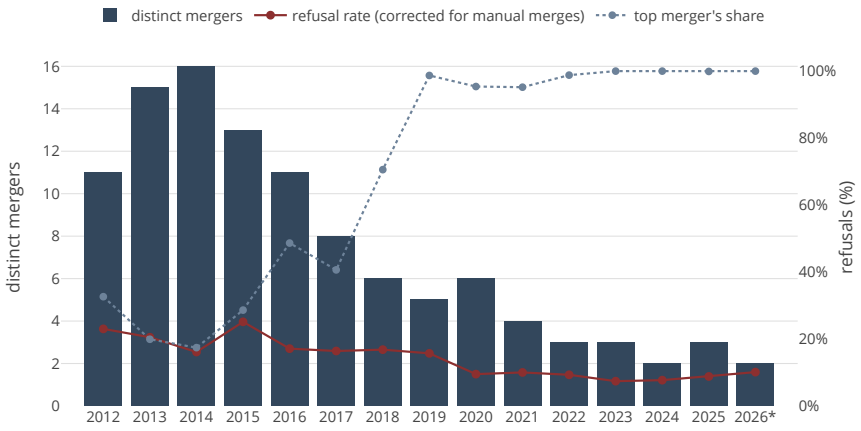
It remains to say why enclosing a feature in a module is the lever, and not an aesthete's preference. The reason is old: Parnas, cited in the chapter "Beneath the interface, dragons", showed that a program must be decomposed along what is likely to change, so that a change stays local. Baldwin and Clark gave its economic translation: modularity creates option value. An enclosed module can be improved, replaced or removed without touching the rest — you are buying the right to change your mind later. A diffuse module withdraws that right: it can no longer be replaced or removed, only endured. Modularity is therefore not a refinement you add when you have the time; it is what decides whether the project will still have choices in 5 years.

An assessment, however, is useless if nobody has the power to act on its conclusion. In the pull-request contribution model, that power exists and it has a holder: the one who merges. He can refuse. The question is whether he does.

The record answers, and the answer is sharp. On the project's repository, for each year, I counted the pull requests merged and the pull requests closed without being merged — hence the refusal rate among decided pull requests only. Once the pull requests closed without merge but integrated by hand are reclassified as acceptances (the closing comment attests to it), it stands as follows:

- 23.0% in 2012,
- 25.1% in 2015,
- 16.8% in 2018,
- 15.7% in 2019,
- 10.0% in 2021,
- 7.4% in 2023,
- between 8 and 10% from 2024 to 2026[10].

The refusal rate has been divided by two or three. And the period when it collapses is exactly the one when the right to merge concentrates: pull requests were merged by some fifteen people in the early 2010s, by two or three since 2022 — a single developer signing each year between 95 and 100% of the merges since 2019 — the full series is in the chapter "Thinking against oneself".



The veto dying out, measured on pull requests (merged_by field — the person who clicks the merge, whatever the mode), 2012-2026: the bars give the number of distinct people who merged pull requests within the year (left scale); the solid line, the share of pull requests actually refused — closed without merge, net of those integrated by hand — relative to decided pull requests only; the dotted line, the top merger's share (right scale). Both scales start at zero and neither curve is rescaled onto another: the coincidence of the declines is in the data, not in the framing. 2026 cut off at August 1.

You would expect the opposite. Power that concentrates is reputed to cut harder; this one cuts less. The veto was not wielded more firmly by a single man: it was wielded less. The chapter "The cargo cult" gave the arithmetic that explains it — the volume merged by this man exceeds several times over what a human being can review. You do not refuse what you have not read.

The project was not, for that matter, unaware of the mechanism. In June 2019, at the public conference for its 10th anniversary, the project's founder and one of the main developers of the first generation drew up the balance sheet of what they themselves called their *laissez-faire* policy, and entered its 4th line item: technical debt — the chapter "The closed circle" quotes the slide. The diagnosis is thus contemporary with the last high year of the curve: 15.7% of pull requests refused that year, just before the collapse to 9-10%. What followed does not correct the diagnosis: it aggravates it.

This rate still has to be set against something, without which it says nothing. The benchmark exists: on the reference corpus of GitHub-hosted projects, the average merge rate stands around 72%, and 13% of pull requests are rejected on technical grounds[11]. The project thus refuses about 2 to 3 times less than the average of the world it lives in. And it is by reading this against what comes in that the figure turns accusatory. A project receiving commissioned contributions — flowing from a roadmap, discussed before being written — can legitimately accept nearly all of them, since the sorting happened upstream. That is not the case here: contributions arrive unconcerted, outside any program, each carrying the idea its author had on his own. A flow nothing has sorted upstream should be sorted at the door, hence refused more, not less. A 9% refusal rate on pull requests nobody asked for does not signal a community of exceptionally sound contributors: it signals a gate without a gatekeeper.

It remains to look at what this refusal to refuse says about the one who does not refuse — for the figures above describe a conduct, not an intention. The chapter "The well-tempered conflict" reports the maintainer's admission: it is sometimes hard to make every decision alone, then take the blame afterwards. There I show what this solitude becomes when it is narrated instead of treated. Here its mechanism must be named, for it has been studied.

Organization science calls it the *romance of leadership*. Meindl, Ehrlich and Dukerich established in 1985 that we spontaneously attribute an organization's results to its leaders, neglecting the other causes — the team, the sector, the circumstances. This is not sycophancy, it is a sense-making shortcut: explaining a performance by its real causes is costly and uncertain, explaining it by one man is immediate[12]. The shortcut works both ways, and that is what gets forgotten: it also serves the one who occupies the center. If everything depends on him, then the state of the project is his merit when things go well, and his burden when they go badly — never a property of the structure.

The narrative and the record can then be placed side by side. The narrative says: I carry alone a load nobody wants. The record says: one person signs between 89 and 100% of the merges, refuses 2 to 3 times less than the average of his milieu, and approves a volume of code that no one can review under those conditions. These two statements do not contradict each other — they describe the same fact under two lights. The first presents it as a sacrifice, the second as an unchecked concentration of power. The difference between them is not psychological, it is political: a

sacrifice calls for gratitude, a concentration calls for accountability.

The admission, for that matter, opened two exits, both of them available. Delegating the power to merge, first — the series shows that it tightened, on the contrary, year after year. Consulting before deciding, second: not gathering reactions after the fact, which the project does abundantly and loudly, but bringing those a decision affects into the decision *before* it is made. That is participatory design, which the chapter "Thinking against oneself" recalls has been documented for 40 years and proved itself with workers who did not program. Neither exit was taken. The one that was taken is the third, the only one that costs nothing: manufacturing a legitimating narrative.

Here I must cite the study that will be held up against me, and say precisely what it measures. Agrawal and his colleagues studied 661 free-software projects and 171 enterprise projects to find out whether "hero" projects — those where 20% of contributors supply 80% of the work — fare worse. Their answer is no. They observe, moreover, that nearly all projects become hero projects as they grow, and conclude that software heroes are "far more common and valuable than suggested by the literature," and that we should learn to retain them[13].

Four things bound that conclusion, three of which come out of the study itself or of its confrontation with the record. First, what it measures: delays — the speed at which tickets and enhancement requests are closed. It measures neither maintainability, nor the erosion of the architecture, nor what

becomes of the project when the hero leaves — that is, precisely the subject of this chapter, and what the bus factor cited above measures in the opposite direction.

Next, and this is more awkward for the objection: on its own indicator, this case fails. The study's entire reassuring reach rests on the presence of heroes not degrading the handling of requests. Yet the chapter "Ship, don't dump" measured the opposite here: human issue closures melted by 53% between 2020 and 2025 — not because there is less to do: the individual yield of those who remain is constant, it is the headcount that melted, and the coverage of the incoming flow recedes from 67 to 60% — the backlog swells again and the stale bot fills the gap — while 47% of reports, as established above, never close. This degradation covers exactly the period when the concentration sharpens — from some fifteen mergers to two. One case does not refute a regularity established across 832 projects; it suspends its use: you cannot invoke, in defense of this house, a study whose reassuring conclusion rests on a variable that, in this house, has collapsed.

Next, the study's favorable result is confined to enterprise projects: in public free-software projects — this one's category — the presence of heroes improves nothing. And the enterprise hero is salaried, mandated, replaceable and accountable to someone: he is a supervised hero, and the supervision is part of the result.

Finally, it is not the concentration of work that I contest. That a single person writes or approves most of the code is nothing scandalous; it is even the common case, and the study has just recalled it. What I contest is that he has to answer to nobody. No

one appointed him to the role, his burden has no term, and his decisions can be challenged before no body. That is the void the narrative comes to fill: it supplies a legitimacy nobody ever granted.

For the narrative of the solitary chief is not a lie, and that is what makes it effective. A narrative its author did not believe would protect no one. This one is sincere, and it is even partially accurate: the load is real, so is the ingratitude. What it accomplishes is a displacement — it converts an organizational failure into a personal quality. Once solitude has become a virtue, sharing it becomes a demotion, and every proposal of structure reads as an indictment. The milieu thereby holds its paradox: the same person can sincerely describe his isolation and sincerely refuse the only remedies that would lift it. You do not get out of this through psychology, you get out through structure — and you understand in passing why maintenance, which is the exact opposite of the exploit, finds no place in a professional imagination built on the figure of the lone man capable of everything, an imagination the sociology of technology has shown to be a gender identity before it is a competence[14].

What non-heroic ambition would be, then, comes down to 4 commitments, none of them spectacular. The first is to put maintenance in the estimate. The ISO/IEC/IEEE 14764 standard classifies maintenance work by moment — proactive or reactive — and by purpose — correcting or improving — and grants preventive maintenance the rank of a category in its own right: correcting latent faults before they manifest in operation[15]. A

named category is a category that can be budgeted, and that someone can be blamed for having omitted.

The second is to demand the enclosed perimeter, that is, to accept a feature only if you can enumerate what it touches, and if that list is short. The question is not "does it work?" but "what will have to be opened to repair it?". A feature whose answer is "just about everything" must be refused as it stands, even good, even already written — especially already written.

The third is to write the rule down. The experienced developer's intuition is not transmitted by osmosis, and the milieu recruits people who are passing through; the acceptance criteria must therefore be written, public and enforceable — which the chapter "The pact" will propose to embed in an institution. This is not a preference, it is a measured result: the survey of the maintainers of 104 popular free-software projects gone obsolete looked for what separates the projects that survive from those that die, and two practices stand out, associated with survival — the existence of written contribution rules, and continuous integration[16]. Structure does not guarantee a project's life, but its absence accompanies its death.

The fourth is to impose a culture and an ethic of work, because they reproduce themselves. Already-tangled code teaches tangling: the contributor who arrives imitates what he reads, and he reads disorder. This is Conway's law from the chapter "The committee will invent nothing" applied to time rather than space — a project reproduces its own past structure. Breaking that is

not done with an annual call to order: it is done by refusing, once, publicly, a contribution that works.

The price of all this must be named, for it is real and it is paid immediately. Refusing a contribution offered for free is discouraging a volunteer. Demanding the enclosed perimeter is lengthening the delay of an awaited feature. Budgeting the preventive is funding work whose product nobody will see — the very definition of what the chapter "Counting the absent" called the avoided bug, which leaves no trace. Non-heroic ambition pays in the visible what it buys in the invisible. That is why it is rare, and that is why it needs a mandate to protect it: no individual will hold alone, year after year, a position whose cost everyone can see and whose benefit no one can.

This book has defended a definition of design: moving an existing situation closer to a preferred one. It carries a constraint of duration that goes unstated. Moving closer, then letting the gap reopen the moment your back is turned, is not having designed: it is having run a demonstration. The only ambition worth having for a working tool is that it still serves when its author has stopped caring about it. There is nothing heroic in that, and that is precisely what makes it hard to obtain from a milieu that rewards only announcements. "Non-heroic" therefore does not mean modest, or cautious, or small in scope: it means rid of the hero model, which is a gender model before it is a work model. The ambition a tool needs does not look like a rescue. It looks like someone who comes back, every day, to where he has already been.

This part of the book has described a collective: what it ships, what it decides, what it costs those who hold it, and what it ought to refuse. It remains to examine the word in whose name it refuses to organize — freedom — and that is the subject of the next part.

---

1. Lehman, M. M. (1980). "Programs, Life Cycles, and Laws of Software Evolution". *Proceedings of the IEEE*, 68(9), p. 1060-1076 https://doi.org/10.1109/PROC.1980.11805 . The 8 laws concern so-called E-type systems — those that execute in the real world and whose use modifies the world they describe, hence all application software. The 3 called upon here — continuing change, increasing complexity, declining quality — are the ones later replications validate most consistently; the laws on self-regulation and organizational stability are more disputed. ↩

2. Lientz, B. P. & Swanson, E. B. (1980). *Software Maintenance Management*. Reading, MA: Addison-Wesley — a survey of 487 organizations, at the origin of the corrective / adaptive / perfective typology; perfective maintenance there exceeds half the total effort. The 60-80% range of lifecycle cost devoted to maintenance is the figure retained by the IEEE Computer Society, corroborated since by Erlikh (2000) and by Boehm's work. The proportions have remained remarkably stable through changes of technology. ↩

3. Star, S. L. & Strauss, A. (1999). "Layers of Silence, Arenas of Voice: The Ecology of Visible and Invisible Work". *Computer*

Supported Cooperative Work, 8(1-2), p. 9-30 https://doi.org/10.1023/A:1008651105359 — invisible work is not work that goes unseen by chance: it is a category produced by the systems of description themselves, which make some tasks countable and others not. Extended, for infrastructures, by Star, S. L. (1999). "The Ethnography of Infrastructure". *American Behavioral Scientist*, 43(3), p. 377-391 https://doi.org/10.1177/00027649921955326 — an infrastructure becomes visible at the moment it breaks down. ↩

4. Archaeology of the Darktable repository (master branch, HEAD 59ed384a1 of July 31, 2026), non-merge commits touching `src/`, authors deduplicated (name and email). Fix = message containing fix, bug, regression, crash, segfault, leak or an issue reference; new feature = add, new, introduce, implement, support, allow, enable; the remainder (refactoring, cleanup, build — 62% of the core's commits) is classified neither fix nor new feature. Patch-up: a fix whose touched files all had the author himself as last modifier, less than 90 days earlier — result identical at the 30- and 180-day thresholds; repairing one's own old code counts as true repair. Figures: 252 developers — 121 with no fix at all (48%), 7 exclusive patch-uppers, 124 true repairers (49%); by tier: 1-2 commits, 146 developers of whom 32% repairers; 3-14, 59 and 56%; 15 and up, 47 and 96%. Volume: 15,599 commits, 3,289 declared fixes (21%), of which 1,758 true repairs (11%) — 1,623 signed by the core (92%). Limits: classification by message, consistent with the heuristics of previous chapters; per-file attribution approximates "introduced by oneself";

"repairing" is measured at the commit, not at the tracker's closure event. ↩

5. The project's GitHub tracker, measured on issues: closure median of 1.3 days and 9th decile of 22 days among those that close, 47% remaining open; to be compared with the 7.4-day median of the peer-help forum's threads. The chapter "The committee will invent nothing" details the method and the share closed by the stale bot. ↩

6. Pierre, A. (2023). "Darktable: crashing into the wall in slow-motion", February 15, 2023. The article also gives, release cycle by release cycle, the ratio of closed issues to merged pull requests: 12%, 12%, 17%, 30%, 46%, then 21%, 15% and 12% — the peak is in the second half of 2021, the collapse follows in 2022-2023. ↩

7. Avelino, G., Passos, L., Hora, A. & Valente, M. T. (2016). "A Novel Approach for Estimating Truck Factors". *IEEE 24th International Conference on Program Comprehension (ICPC)* https://doi.org/10.1109/ICPC.2016.7503718 — the *bus factor* is the minimal number of developers whose departure would leave the project unable to continue. On a corpus of 1,932 projects, 16% saw all of their key developers leave, and development was taken over by others in only 41% of those cases. ↩

8. ISO/IEC 25010, *Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model* (2023 revision) https://www.iso.org/standard/78176.html. Maintainability is defined there as the degree of

effectiveness with which a product can be modified, and broken down into modularity, reusability, analyzability, modifiability and testability. ↩

9. MacCormack, A., Rusnak, J. & Baldwin, C. Y. (2006). "Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code". *Management Science*, 52(7), p. 1015-1030 https://doi.org/10.1287/mnsc.1060.0552 — propagation cost is the average proportion of a system's elements affected by the modification of one element drawn at random, computed on the design structure matrix. The Linux / Mozilla comparison and the drop of Mozilla's propagation cost after its overhaul serve as controls. The economic reading of modularity as option value is developed in Baldwin, C. Y. & Clark, K. B. (2000). *Design Rules, Vol. 1: The Power of Modularity*. Cambridge, MA: MIT Press. ↩

10. Pull requests of the Darktable-org/Darktable repository, queried through the GitHub API (`type:pr is:merged` against `type:pr is:unmerged is:closed`, by year of creation; 2016-2026 read on July 24, 2026, 2012-2015 on August 1). **Correction of false closures**: a pull request can be closed without merge because the merger integrated its content by hand (rebase or rework outside the branch); scanning the closing comments of the 1,583 pull requests closed without merge (patterns "manually merged", "merged manually", "cherry-picked", "rebased and merged", etc.) identifies 211 of them, concentrated where they distort the most: 32 in 2018, 30 in 2019, 54 in 2020. These pull requests

are reclassified as acceptances; the rates in the body and in the chart are the corrected rates. Raw series, merged / closed without merge (manual merges): 92/30 (2) in 2012, 227/61 (2) in 2013, 318/67 (5) in 2014, 182/73 (9) in 2015 — the plural era refuses within the orders of magnitude of the literature —, then 238/55 (5) in 2016, 161/34 (2) in 2017, 246/88 (32) in 2018, 723/170 (30) in 2019, 1,635/231 (54) in 2020, 1,485/181 (14) in 2021, 1,163/133 (13) in 2022, 1,495/139 (18) in 2023, 1,083/106 (14) in 2024, 864/91 (7) in 2025, 821/97 (4) in 2026. That is, after correction, 10,944 pull requests accepted against 1,345 actually refused — 10.9% refusal over fifteen years. Two caveats: the rate covers decided pull requests only, those still open being excluded; and closure without merge does not distinguish a motivated refusal from abandonment by the author or from automatic closure. These caveats cut in the same direction and do not change the order of magnitude of the trend; the manual-merge correction, which cuts the other way, has been made. ↩

11. Gousios, G., Pinzger, M. & van Deursen, A. (2014). "An Exploratory Study of the Pull-Based Software Development Model". *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, p. 345-355 https://doi.org/10.1145/2568225.2568260 — on the GHTorrent corpus, the average merge rate stands around 72%, and about 13% of pull requests are rejected on technical grounds. The bases are not strictly identical to the measure above, which excludes pull requests still open; the comparison holds as an order of magnitude, not as an equality of method. ↩

12. Meindl, J. R., Ehrlich, S. B. & Dukerich, J. M. (1985). "The Romance of Leadership". *Administrative Science Quarterly*, 30(1), p. 78-102 https://doi.org/10.2307/2392813 — the authors show that an organization's performance is spontaneously attributed to its leaders at the expense of other factors (team, sector, conjuncture), and analyze this bias as a sense-making shortcut: real causality is complex, causality through one man is immediately tellable. The notion has been taken up and extended since, notably in the special issue of *The Leadership Quarterly* devoted to his memory (2011). ↩

13. Agrawal, A., Rahman, A., Krishna, R., Sobran, A. & Menzies, T. (2018). "We Don't Need Another Hero? The Impact of 'Heroes' on Software Development". *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, p. 245-253 https://doi.org/10.1145/3183519.3183527 ; preprint https://arxiv.org/abs/1710.09055 . The authors define a "hero" project by the 80/20 rule, study 661 free-software projects and 171 enterprise projects, and conclude: "Software heroes are far more common and valuable than suggested by the literature, particularly for medium to large Enterprise developments." Their outcome variables are processing delays (closure of defects and enhancement requests); they note themselves that the favorable effect is observed only on enterprise projects, the presence of heroes not improving the processing rate of enhancement requests in public free-software projects. ↩

14. Faulkner, W. (2007). “Nuts and Bolts and People: Gender-Troubled Engineering Identities”. *Social Studies of Science*, 37(3), p. 331-356 https://doi.org/10.1177/0306312706072175 — the engineer’s professional identity is built on a valuation of the “purely technical” at the expense of the relational and the organizational, a divide itself gendered, and one that makes the work of coordination and upkeep barely legible — indeed barely honorable. ↩

15. ISO/IEC/IEEE 14764:2022, *Software engineering — Software life cycle processes — Maintenance* https://www.iso.org/standard/80710.html. Categories of work are defined there by crossing the moment of the change (proactive or reactive) with its purpose (correction or enhancement), which yields the 4 types: corrective, adaptive, perfective and preventive — the last designating the correction of latent faults before they manifest in operation. ↩

16. Coelho, J. & Valente, M. T. (2017). “Why Modern Open Source Projects Fail”. *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, p. 186-196 https://doi.org/10.1145/3106237.3106246 — a survey of the maintainers of 104 popular GitHub projects gone obsolete, yielding 9 failure reasons split between the team (lack of time, lack of interest, conflicts between developers) and the project itself (obsolescence, competition). The authors note that the adoption of contribution rules and of continuous integration shows a strong association with a project’s

failure or success. Association measured, not causality demonstrated. ↩

Part V — Freedom and its conditions

24. Freedom in a void

“The code is free, you can just fix it yourself.” Every free-software user has been handed that line one day, in reply to a reported defect. It is perfectly accurate, and it serves no purpose other than intimidation. The user who has just received it cannot program, does not have 10 years to learn C, and knows it; the one who says it knows it too. The sentence does not inform: it closes. It converts a freedom into a dismissal. That conversion is not a forum accident — it is lodged at the heart of the doctrine of free software. Taking it apart is not an exercise in style: as long as it holds, free software will remain a club, instead of the public service it claims to be.

Free software rests on 4 freedoms: to run the program, to study how it works, to redistribute copies, to change it. They are fine freedoms, I owe them what I have built, and this book would not exist without them: written on Linux, in Visual Studio Code, published on a website built with Hugo, on a Linux server. But their form, first: all four say what nobody has the right to *forbid* you. Open doors — nothing more. None says a word about what it costs to walk through the door: knowing how to read C, having free time, a development machine, years of practice. Their audience, now: 3 of these freedoms — studying, changing,

redistributing code — can only be exercised by someone who knows how to program, and less than 6% of the world's population can write code[1]. So these freedoms must be called by their name: they are **engineer's freedoms**. A free license lifts intellectual property from the code — an operation engineers know how to profit from, and they are just about the only ones. For the remaining 94%, the door is open and the room out of reach; the chapter "Counting the absent" gave the rest of the figures — half the world cannot install software. A freedom that benefits only those who already possess the skill is not emancipation: it is a privilege presented as a right.

This objection is not mine, and it is older than software. Isaiah Berlin distinguished *negative* liberty — the absence of interference: no one blocks me — from *positive* liberty — the effective capacity to do[2]. The 4 freedoms are purely negative, and the modern vocabulary of this critique belongs to Amartya Sen: what counts is not the formal right but the *capability*, what a person is actually able to do and to be[3]. Proclaiming that everyone is free to buy bread changes nothing for those who cannot pay for it; proclaiming that everyone is free to fix the code changes nothing for those who cannot program. Martha Nussbaum named exactly what is missing: *combined capabilities*, the conjunction of an internal aptitude and the external conditions of its exercise[4]. A license supplies the permission; it supplies neither the aptitude, nor the time, nor the usable tool. Free software preaches a freedom defined outside any context and any structure — freedom in a void.

Virginia Held had made the diagnosis one floor above software: contract theories, she wrote, err in "taking the ideal circumstances of an adult, independent head of a household as paradigmatic and ignoring all the rest" — that "every person starts out as a child dependent on those providing us care," and that the independence we credit ourselves with rests on a network of relations that makes it possible[5]. A license is contract thinking, and it inherits the blind spot by construction: that the four freedoms serve imaginary equals and leave real dependence at the door is not a drafting accident — it is what the contractual form produces everywhere it is employed.

And the user, in this transaction? He is excluded by the very text. Every free license states it in capitals: the program is provided AS IS, WITHOUT WARRANTY OF ANY KIND, not even the implied warranty of fitness for any purpose whatsoever. The user has no right but to run the code as it stands — no right to support, no right to fixes, still less to seeing the tool fit his needs. If help is provided, it is at the developers' discretion, and eternal gratitude is expected in return. Legal caution, fine. But set that clause beside the pitch that accompanies the same software on its web page — professional quality, a credible alternative to commercial tools, join us. The user is drawn in on political and ethical ground — freedom, data sovereignty; and when he comes back to report that the tool makes him suffer, he is met on economic ground: volunteer work, no money, no liability — you could just have fixed it yourself. The entry pitch spoke only of freedom and, between the lines, of emancipation; nowhere did it say that by installing, you renounced the right to find the tool bad. The word

“professional” was uttered, and the promise not kept: somewhere in there is a deceptive incoherence. Freedom is on display in the shop window; the non-warranty is in capitals, but hidden in the source code.

Where does such a blind spot come from? From the movement’s genealogy, and it can still be read in its reflexes. Free software is the child of hacker culture — individualist by nature, compatible with capitalism by proxy, via technophilia and technological solutionism. In that culture, *sharing as is* — one’s code, one’s hacks — was confused with *taking care of* — other people’s needs, limits and difficulties — and the confusion was wrapped in quasi-hagiographic narratives. When code is all you know how to make, code is the solution to everything. Yet at the moment the movement is born, in the 1980s, the fraction of humanity that owns a personal computer is minuscule; the freedom of code is the product of that privilege, thought by it and for it. Privileged minorities are always blind to their privileges — that is even how you recognize them. Free software is a freedom for those who have the means to afford it; and the fact that the label carries no currency symbol makes the thing vicious, for the price exists — in skill, in time, in machines — and not displaying it lets you claim there is none.

We must dwell on this confusion, for it is the subject of the entire book and the moment has come to say it plainly. Listening before deciding, characterizing the need before the solution, verifying reception instead of presuming it, maintaining instead of dumping, answering for what you ship: the first 4 parts have described, method after method, what *taking care* means in a

technical trade. They did so without uttering the word, because the gestures had to be established first; the conclusion will gather them under their true name. We will then see that free software never lacked generosity — it lacked everything that, in taking care, is not the sharing of a finished product.

The same genealogy explains why the movement never thought about work. Its implicit frame is the individualist free market of freelancers and consultants: everyone is free to launch their project, to fork everyone else's, and social Darwinism will sort it out. Fine — but who pays for the work? Capitalism only gives work value if its product sells, and value follows scarcity: a dematerialized product, downloadable for free ad infinitum, has zero market value by construction. The movement thus invented a production structure communist by nature — pooling of work and product — and embedded it in a capitalist ideological frame that deprives it of any means of financing its own work. The funding strategies it has been seen trying for 40 years are all precarious because they inherit this contradiction, and the chapter "The invisible worker" showed who mops up the deficit: the invisible worker, in unpaid hours. I will come back to it in the last chapter of this part, for this contradiction has a way out — but it requires finally thinking what the movement never wanted to think: the social structure of work. The philosophy of technology had given warning, though. Artifacts have politics, wrote Langdon Winner: the way a technology is born and organized embodies power relations that no proclaimed neutrality erases[6]. Andrew Feenberg draws from it the consequence free software missed: the democratization that

matters is not that of the finished product, but that of the *design process* and the conditions of work — and that is precisely where the movement, having freed the license, freed nothing at all[7].

The orthodox objection founds the whole movement. The license, it replies, never promised anything else: the 4 freedoms protect against proprietary enclosure, period; demanding usability or well-being from them is confusing a freedom with a service. And the essential must be conceded: wresting software from exclusive ownership was a real fight for emancipation, the foundation of everything that followed — including my own work. But the argument proves too much. If formal freedom sufficed, property rights would abolish poverty and the right to vote would make democracy; nobody holds that, precisely because the distinction between the right and the means of the right is settled everywhere else. The public hospital is the familiar demonstration: it is not the "competitor" of the private clinic, it is of another nature — a public service exists to give people the means to exercise their rights without means-testing, because a freedom remains theoretical as long as you lack the means to make use of it. That is the definition of a public service, and it is word for word what Sen says of freedom. The answer is therefore not to tear down the 4 freedoms — they are a necessary floor — it is to refuse them the status of ceiling. Opening the door is indispensable; it does not excuse you from building the staircase. And the staircase — training, documentation, the transparent tool of the chapter "The transparent tool", the working conditions of the chapter "The invisible worker" — is a matter of design and organization, not of law. It is work, someone has to do it, and the

license is silent on who. Thirty years sufficed to free the code; it remains to free the one who uses it, and no clause will do that.

The same illusion has an internal face, turned toward those who produce. In the name of the same value, projects organize themselves as freedom without structure: no hierarchy, no procedure, no designated lead. The chapter "The committee will invent nothing" gave Freeman's verdict — the absence of structure does not abolish power, it makes it invisible — and it must now be pushed to its political consequence, for the symmetry with the user is exact. The producer's freedom, as the milieu practices it, is also purely negative: free of bosses, free of procedures, free of obligations. Yet what a collective needs in order to govern itself is a positive liberty, in the same Berlin's sense: the capacity to *bind itself* — to give itself rules, make commitments, designate leads and be able to remove them. A group that cannot bind itself can promise nothing, neither to its members nor to its users; it does not decide, it drifts — and the chapter "Ship, don't dump" measured what drifting means.

The power that fills this void has, moreover, a property that instituted power never has: it is irrevocable. Freeman put it dead straight: "Their power was not given to them; it cannot be taken away." A designated lead can be removed; an informal elite cannot — no mandate to withdraw, no election to lose, no body to appeal to. There is not even an address where to lodge the grievance: ask who decides, and you will be told "the community," and the chapter "Thinking against oneself" counted what "the community" means in practice — two mergers, and for the essential a single one since 2019. The only sanction left is exit:

leaving, or forking — precisely the gesture the milieu treats as betrayal. And the medal must be turned over to see its other face: a power without a mandate cannot be revoked, but neither can it be *obligated*. The license declared that nobody owed anybody anything; the absence of structure guarantees that nobody answers for anything. "If the movement continues deliberately to not select who shall exercise power," Freeman concluded, "it does not thereby abolish power. All it does is abdicate the right to demand that those who do exercise power and influence be responsible for it"[8]. Rights concentrated without the duties that go with them — history has names for that regime, and "horizontality" is not one of them.

Freeman, for that matter, had not stopped at the diagnosis, and it is the half of her essay the milieu never cites: it ended with "principles of democratic structuring" — delegating defined authority, for defined tasks, by democratic procedures; making delegates responsible to those who chose them; distributing authority as widely as reasonable; rotating tasks; allocating tasks along rational criteria; diffusing information to everyone, for "information is power"; ensuring equal access to resources[8]. In other words: the institutional conditions of collective freedom, written as early as 1972 by the movement that had tried horizontality before us and paid to see. Free software inherited the refusal of bosses; it did not inherit the instruction manual.

Freedom in a void thus produces twin voids. On the user's side: a right of access without the capacity to use it. On the producer's side: a facade of horizontality under which informal elites decide without mandate and without accountability. In both cases, the

word *freedom* covers exactly what should be looked at — a skill that is missing, a power that hides — and in both cases, what it covers is a matter of conditions, not permissions: capabilities to build on the usage side, institutions to build on the production side. The license is silent on both.

There lies the underlying flaw: the movement defined freedom as the absence of interference, and believed the job done. Everything that decides whether a freedom is exercised or remains a dead letter — the user's capacity, the worker's conditions, the collective's structure — stayed out of frame. That out-of-frame is not empty for everyone: it is occupied by those who already had the skill, the time and the machines — the 6%, and among them a still narrower fraction. Who are they, exactly? I put the question to my own users, twice, with numbers to back it. The answer occupies the next chapter, and it explains a great deal.

---

1. Estimate based on International Telecommunication Union data on digital skills (see the chapter "Counting the absent"); I documented it in the analysis of my first user survey ("Who are the Darktable users in 2020?", eng.aurelienpierre.com, 2023, § "Ability to interact with the computer") https://eng.aurelienpierre.com/2023/01/who-are-the-Darktable-users-in-2020/. ↩

2. Berlin, I. (1958). *Two Concepts of Liberty*. Oxford: Clarendon Press; reprinted in *Four Essays on Liberty*, Oxford University Press, 1969 https://philpapers.org/rec/BERFEO . I borrow

Berlin's taxonomy, not his preference: he defended negative liberty. ↩

3. Sen, A. (1979). "Equality of What?". Tanner Lecture, Stanford University https://tannerlectures.org/lectures/equality-of-what/ ; developed in *Development as Freedom*, New York: Knopf, 1999 https://www.worldcat.org/oclc/41404591 . ↩

4. Nussbaum, M. C. (2011). *Creating Capabilities: The Human Development Approach*. Cambridge, MA: Belknap Press https://doi.org/10.4159/harvard.9780674061200 . ↩

5. Held, V. (2006). *The Ethics of Care: Personal, Political, and Global*. New York: Oxford University Press, pp. 81 and 13-14 https://doi.org/10.1093/0195180992.001.0001 : "taking the ideal circumstances of an adult, independent head of a household as paradigmatic and ignoring all the rest"; "Every person starts out as a child dependent on those providing us care". ↩

6. Winner, L. (1980). "Do Artifacts Have Politics?". *Daedalus*, 109(1), 121-136 https://www.jstor.org/stable/20024652 — technical artifacts embody power relations; some technologies are political by their very form. ↩

7. Feenberg, A. (1999). *Questioning Technology*. London: Routledge https://doi.org/10.4324/9780203022313 — "democratic rationalization": democratizing a technology means opening its design process, not merely distributing its product. ↩

8. Freeman, J. (1972). “The Tyranny of Structurelessness”. *Berkeley Journal of Sociology*, 17, 151-164 https://www.jofreeman.com/joreen/tyranny.htm; see the chapter “The committee will invent nothing” for the diagnosis. Verbatims: “Their power was not given to them; it cannot be taken away”; “If the movement continues deliberately to not select who shall exercise power, it does not thereby abolish power. All it does is abdicate the right to demand that those who do exercise power and influence be responsible for it”; “Information is power”. The 7 principles close the essay under the title “Principles of Democratic Structuring”; the body of the text condenses them following the order and wording of the original. ↩ ↩

25. What insularity produces

Who makes free software? Lacking a census, I measured what I had at hand: my own users, surveyed twice with the method of the chapter "Counting the absent" — 1,101 responses from 59 countries in 2020, 1,253 in 2022, anonymized data and processing code published with the articles, so that anyone can check[1]. Measuring the users means measuring the pool from which contributors come: among the 2022 respondents, 65 declared contributing to the project, and the sample will say in a moment what that pool looks like. The measurement goes beyond the case: any project with a similar composition will produce the same blind spots, and that is why we count them. The archetype that comes out, as I published it: a man of about 50, between 25 and 65, holding an engineering degree or equivalent. By itself, this archetype covers 55% of the user base. One user in two can program — 46% in 2020, 55% in 2022, the filter is tightening — that is, 7 to 8 times the proportion in the general population; the leading field of study, on its own, is computer science and electrical engineering, at 45%. Women: 2% — 25 people out of 1,243, and two among the 65 contributors. Yet the trade this tool serves is majority-female: about half of American professional photographers are women, as are 60 to 75% of photography graduates. Between the trade and its free tool, the share of women is divided by 25. This is not an impression, it is a measurement — and I had drawn the conclusion in the article: free software does not discriminate on money, race, age or

religion; it discriminates on computer literacy, and apparently on gender.

Where does this composition come from? From the previous chapter: the freedom of free software has a price without a currency symbol — skill, time, machines — and you have to look at who can pay it. The answer fits in a double subsidy. The first is class: the reference survey of free-software developers found 97.5% men, mostly employed in IT, and as the first motive for contributing intellectual stimulation — play[2]. And play is not just a motive for entry: it is a criterion of governance. On the Darktable project's chat room, a moderator reports having proposed objective criteria for accepting and refusing contributions — shot down, among other reasons, as "impacting developer fun"[3]. The figure is no artifact of a single survey: the major censuses of free software all yield the same crushing masculinity — on the order of 1% women in the earliest studies, 3% in the survey of a major code forge, around 10% at the top of the recent estimates[4]. The typical contributor has an engineer's salary that pays for the machine, the high-end camera and the available evenings; photography is for him a hobby with no obligation of result — you can throw away the failures, work in small batches, take a break whenever you like — and his tolerance for interface defects is above average, since the terminal does not scare him. The second subsidy is gendered, and it too can be measured: in European countries, women provide about 260 minutes of unpaid domestic work per day, men 140; in France, the Time Use survey attributes 64 to 72% of domestic hours to women[5]. Sociology has had a name for this since 1989: the

second shift, and its corollary, the leisure gap — male free time is longer and less interrupted. The free evening that funds free software is therefore subsidized twice: by the salary, and by a woman doing the shopping while the man of the house plays at the computer. And that scene is timestamped. The chapter "The committee will invent nothing" plotted the hourly distribution of the project's commits, in each author's local time: the work concentrates between 5 p.m. and midnight, and more than one commit in three is signed between 5 and 10 p.m. — the hour of errands, dinner and putting the children to bed. The heartbeat of the repository is the exact negative of the domestic schedule: what the record of the code registers without knowing it is the hour at which someone else is running the household. It is a structure, and you cannot understand what free software produces without naming it.

This double subsidy does not merely fund the leisure: it turns it into class contempt. The contributor practices photography without depending on it, and he knows it — "I'm glad I don't have to make a living from it" is a sentence you hear in these circles, spoken without embarrassment. Hence an inversion of values: the professional, the one for whom the image is a livelihood under constraints of deadline, volume and client, becomes in his eyes the poor guy who needs software that works, while he himself enjoys the luxury of the tool that resists, of the terminal, of the endless configuration — badges of membership in the caste of those who know. Lightroom becomes the Muggles' software; free software, that of the real wizards — and the expulsion is pronounced without detour: on the project's chat

room, to someone describing the difficulty of educating lost users, a regular replied that "such users should stick to Adobe products please"[6]. The paradox is bitter: in the name of emancipation, free software is in practice *less* accessible to the non-programmer than the commercial software it fights, whose interface, for its part, was designed to be sold to people without technical skills. And that barrier is invisible from the milieu that erects it, for a reason of pure social geometry: those it stops never enter the room. They will not file a bug report, will not come to the forum to tell why they gave up — for them, computing is a counter you endure, not a playground — and their absence makes no noise. This is the social side of the chapter "Counting the absent". The idea that, if your pastime is someone else's real trade, you will never be truly good at this game does not cross their minds. Surplus time, in Veblen, was already the marker of a class proving its rank by devoting itself to unproductive occupations and despising necessary labor[7]; the gesture is intact, transposed to code. And the history of technology offers a troubling precedent: Ruth Cowan showed that the household appliances of the 20th century, supposed to free women from domestic work, instead redistributed and intensified its load[8]. The cause I read in it — they had been designed by people exempt from that work, for an idea of the home that was not that of those who lived in it — is the same as here: designers exempt from the need they claim to serve were already producing, a century ago, the same result. Free software invents nothing; it replays.

For a room so composed does not produce just anything: it produces what it is. The sociology of technology has named the mechanism: *I-methodology* — the designer who, without even thinking about it, takes himself for the user[9]. The textbook case of the literature is an all-male team convinced it is designing for "everybody," and which configures that everybody in its own image[10]. I do not need to borrow the textbook case: mine is public. In June 2026, on the community forum, the umpteenth discussion of Darktable's interface — opened on the 22nd, closed on the 26th, 246 messages in 4 days, the usual fate of these threads — produced a confession in due form. The project's release manager writes there, in so many words: "So, let me voice it: I do like a lot the way the UI is"; then, two messages later: "Darktable devs are also photographers. And I suppose we can say: The Darktable devs like the way the UI is"; and when a user offers "I think you could say 'Darktable is created for nerds by nerds'", the correction falls: "No you can not. The project has chosen for good reasons: 'Darktable is created for photographers, by photographers'"[11]. Everything is there, without any of it having to be dragged out: personal taste erected into a criterion, the team taken as a sample of the users, and I-methodology promoted to official motto — *by* photographers, *for* the photographers, that is, by us, for us. A regular of the same forum had said it more crudely years before: the software is made because its developers wanted a piece of software that suits them. The chapter "Listening without obeying" established why the designer cannot take himself for the user; here is what happens when an entire community does it with pride.

Such a room does not reopen on its own: homogeneity, once installed, reproduces itself. The chapter "Counting the absent" gave its first spring, survivorship bias: those who were put off are no longer there to say why they left, so that the room hears, on a loop, only people who resemble it and comfort it. It is an echo chamber. The second spring is the simplest: including takes a conscious effort, and a room that has made its taste its motto will not supply it. Any community mixing photography and free software will tilt by default toward the well-off, educated, English-speaking man at ease with computers, and it will take deliberate, daily work to reach out to those this filter excludes, on pain of feeding exactly the survivorship bias. A community that, instead of supplying that work, celebrates its closure as an identity, locks its composition and, with it, the list of needs it is capable of noticing.

To these two springs, my case study adds a third, more destructive than the first two: the prestige economy there is indexed on production, never on knowledge. You earn your stripes by publishing software — an amusing module, a toy tool the others can try that very evening — and there are no stripes for what you *know* and what that actually improves in the tool. Yet most of what has been announced on that forum in image processing rests on faulty color science and betrays a deep misunderstanding of the underlying theory — which prevents neither the pleasure of use, nor the congratulations, nor the reputation that goes with them. The forum applauds what runs, not what is right; between the true and the fun, the room has chosen, and it does not know it has chosen.

The inverse selection follows mechanically, and it has names. Whoever tries to teach, primary sources in hand, must contradict one by one programmers and amateurs full of opinions on art and color, with nothing substantial to back them; and the asymmetry of that work has its aphorism, Brandolini's law: refuting bullshit costs an order of magnitude more than producing it[12]. Three people who were doing precisely that work of education left that forum for related reasons: Elle Stone, who had long worked on color profiles, Troy Sobotka, whose Blender filmic is the origin of mine, and me. My weariness can be dated, once again. In January 2022 — 4 months before the breaking point the chapter "The closed circle" dated to May 7 — I open a thread to re-explain the basics for the umpteenth time, and the opening of the message already says the attrition: people who keep ignoring the advice come back to complain "why-the-fuck-did-you-change-display-referred-that-got-me-faster-results"[13]. Because I was the programmer behind the functions, I could never begin by explaining how to use them: I first had to undo what people believed they knew, which was at best inferences about the nature and workings of the tools based on their behavior as visible from the interface. These places do not favor intellectual rigor; they favor people who have free time. And one day, the balance sheet draws itself: this forum brought me nothing — not one joy, not one instructive conversation — only time lost trying to save the users from themselves, and from the pack of preachers who, in trying to help, spread false information that wastes everyone's time.

It remains to see what this room can*not* produce, and this is where composition becomes technical destiny. A room of fifty-something engineers who came to play selects implicit values: technical play over care, novelty over maintenance, the exciting over the useful. The ethics of *care* has described the exact mechanism: caring begins with *noticing that a need exists* — and that is precisely the phase that fails in a room from which the bearers of the need are absent; Tronto calls "privileged irresponsibility" the luxury of not having to notice[14]. This ethics is not summoned here in passing: it is the frame in which the conclusion will reread the whole book, and one can already note what makes it embarrassing for the milieu — its first phase, the most elementary of all, is precisely the one a homogeneous room cannot cross. And the historians of technology have shown that the cult of novelty devalues maintenance by the same cultural gesture that devalues domestic work — the comparison is not mine, it is in their text: the obsession with technological novelty "obscures all of the labour, including housework, that women, disproportionately, do"[15]. Darktable's measured priorities are the numbered demonstration: MIDI controllers — owned by 10% of users, used by 2% — earned a complete overhaul of the input system, because that is an engineer's toy; graphics tablets — owned by 37%, the standard tool of the trade — have support so deficient that only 6% use them, and it has never been a priority. The room's toy beat the trade's tool, in the numbers.

And this dialogue of the deaf must not be believed recent, nor tied to one person: it is structural, and it has a dated archival exhibit. In March 2016 — 6 years before the conflicts this book has

recounted — a user opened on the community forum a thread titled “Does the world need another incomplete photo editor?”. His request was pure needs-level, on behalf of absent users: improve the stability and speed of what exists *without adding options*, make the software accessible to normal people. The replies, reread today, form an almost perfect catalog of the registers of this chapter. The register of play: “People write (and share) things that scratch their own itch”; we are not paid — “it’s inappropriate to tell them to work on something more useful”; “I do Filmulator for me. I give it away because I can, not because I think you should use it”. The register of the inverted debt: “that’s the implicit contract that comes with free software” — you receive the work of others and repay with your time; “Freeloaders are a burden and not an asset”. The register of dismissal: remarks “most of which are based on ignorance”, “how are you offering to help out?”. And when a developer — for there was one, and he is the counter-example that honors the thread — argued that whoever gives away software owes its users at least not to harm them, he was answered with *the text of the license*: “THERE IS NO WARRANTY FOR THE PROGRAM” — the legal clause brandished as the answer to a moral question. Another developer ended up naming what had just happened: the original question — “how could we better utilize developers to more completely meet the needs of typical users?” — had been devoured by the debate on what developers owe or do not owe; the thread was closed the next day by a moderator, on a joke, the question left unanswered[16]. Everything is there, at 6 years’ distance: the two levels of discussion — the need against the itch —, the request for care perceived as an aggression, and a

“community” that, summoned to choose between serving needs and remaining a circle of interests, chooses the circle without even perceiving that there was a choice.

Yet this thread must be read for what it is: talk. Both camps existed in it — that is to the thread’s credit — but they existed only in discourse, and it is the following decade that settled between them, by the only judge that counts: the energy actually invested. The camp of the need produced no structure; the camp of the itch produced the status quo, and the status quo has a slope. Ten years later, the same complaints return, almost word for word — stability, option inflation, inaccessibility to normal people; mine have joined them, and the chapter “The closed circle” will tell what it cost. The piling-up of features has continued — the chapter “In praise of the simple” counted it. And the composition of the user base, measured at the head of this chapter, gives the direction of the movement: between 2020 and 2022, the share of programmers among users went from 46 to 55% — the filter did not loosen, it tightened. There, then, is the complete picture: a state of mind revealed in 2016 in a discussion thread, then verified as a trajectory over 10 years of facts. For want of a structure to carry the other camp, the talk remained talk, and the slope did the rest.

And it must be said who applauds, for the supply would not persist without the demand. Browse the announcement threads of a new release: the enthusiasm there is immediate, massive — and undifferentiated. *Every* novelty is acclaimed, brand-new module, additional option, experimental mode, without the only question that counts in a professional tool ever being asked: does

this make my photographs better? It is the technophile bias in its pure state — technical novelty as a value in itself, inherited from the genealogy of the chapter "Freedom in a void" — and it works as an economy of acclamation: the developer who ships an exciting feature is applauded that very evening; the one who repairs a cache for 4 years makes no announcement. The chapter "Code is a liability" had noted that users judge the health of a project by its agitation; here is the mechanism in full: a room that applauds everything that moves trains its developers to make anything move. The validation criteria of the chapter "Stating the problem" — the task, the time, the result — never enter this reward loop; the applause, for its part, is instantaneous. You get exactly what you reward, and this room rewards the new, never the right. As for professional photographers, they are not on GitHub or on IRC, where the decisions are made — and if they do not know what IRC is, that is precisely my point. Who, then, can say what they need to be productive? Nobody. But who wants to know? Nobody either: since they do not use our tools, that is proof enough that they are button-pushers — and the prophecy fulfills itself, chapter "Counting the absent", social version.

Hence this chapter's thesis: **what free software produces is everything such an environment can produce — nothing, in this environment, supplies a reason to produce anything else.** Where counter-forces exist — users who carry weight, designers different from the typical designer —, I-methodology recedes, and documented cases are known[17]. But in their absence, it operates by default, and the environment delivers the product its composition selects. It is the sociological twin of

Conway's law from the chapter "The committee will invent nothing": software copies the communication structure of the organization that builds it — it also copies its social composition. A trade tool produced by a room without the trade, an interface for terminal experts produced by terminal experts, maintenance devalued by a culture that never had to do the housework: none of this is an accident.

The objection protects something true: it is their free time, they owe nobody anything; whoever scratches his personal itch answers only to his itch. For the personal project, it is entirely right — the weekend repository is accountable to no one, and the hobby ethic is fully legitimate there. But two things change the deal. The first is status: when a piece of software becomes the working tool of hundreds of thousands of people, it changes category — it becomes infrastructure, and you do not judge an infrastructure by the yardstick of its makers' leisure; the chapter "Freedom in a void" showed that it is the project itself that claims this status, by selling itself as a professional alternative. The second is deeper: the itch is not an accident. Scratching your itch is innocent; what is not innocent is the composition of the room that decides *which itches exist*. When the room is homogeneous, Raymond's itch management becomes a social filter: the needs of the absent itch nobody, so they do not exist. The answer is not to demand that volunteers work harder — it is to stop celebrating the closure of the room as a motto, and to ask the question the last chapter of this part will face: how you open the door.

Before getting there, it remains to describe what happens when the absent insist all the same — when an off-profile user comes to

say that the tool makes him suffer, and what the room answers him. That is the subject of the next chapter.

---

1. "Who are the Darktable users in 2020?" (n = 1,101, 59 countries) https://eng.aurelienpierre.com/2023/01/who-are-the-Darktable-users-in-2020/ and "Who are the Darktable users in 2022?" (n = 1,253) https://eng.aurelienpierre.com/2026/07/who-are-the-Darktable-users-in-2022/, eng.aurelienpierre.com — anonymized data and processing code downloadable with the articles. The archetype and its 55% coverage are the published conclusion of the 2020 survey; the 2.0% of women (25/1,243), the 54.6% of programmers and the 44.9% of computer science/electrical engineering are measured in 2022. Baselines: US Census, Pew, ITU, StatCounter. Acknowledged sampling caveat: 68% of the 2022 respondents recruited through Darktable channels — the most favorable sample possible for the project. ↩

2. Lakhani, K. R. & Wolf, R. G. (2005). "Why Hackers Do What They Do", in Feller, J. *et al.* (eds.), *Perspectives on Free and Open Source Software*. Cambridge, MA: MIT Press https://doi.org/10.7551/mitpress/5326.003.0005 — n = 684: 97.5% men, 58% employed in IT, first motivation "intellectually stimulating" (44.9%), 73% describing flow states. ↩

3. Same log, January 18, 2023, the moderator's verbatim: "I've also suggested having some gates up for what to accept and

what to turn away, and that was shot down for a variety of reasons, including 'impacting developer fun'". The reporter is here a critical witness — he was the one proposing the criteria; the phrase "developer fun" belongs to the developers who refused them. The original discussion has not been found — neither on the GitHub repository, nor on the forum, nor in the period covered by the chat-room log: the refusal took place outside the consultable archives, and the moderator's testimony remains its only exhibit. The criterion invoked, though, is not isolated: the project's 10-year conference, in 2019, already stated its only rule: "if there is a rule, it's don't stop anyone from coding when they're having fun" — see the chapter "The closed circle". ↩

4. Ghosh, R. *et al.* (2002). *Free/Libre and Open Source Software: Survey and Study — FLOSS Final Report.* University of Maastricht https://www.researchgate.net/publication/264799695_The_freelibre_and_open_source_software_survey_and_study-FLOSS_final_report — 1.1% women; GitHub (2017), *Open Source Survey* https://opensourcesurvey.org/2017/ — 95% men, 3% women; Trinkenreich, B. *et al.* (2022). "Women's Participation in Open Source Software: A Survey of the Literature". *ACM TOSEM*, 31(4) https://doi.org/10.1145/3510460, 1-37 — about 9.8%. ↩

5. Hochschild, A. R. & Machung, A. (1989). *The Second Shift.* New York: Viking https://search.worldcat.org/title/18588639. Current figures: OECD (2025), synthesis across 23 European countries — 262 minutes per day of unpaid work for women,

141 for men; INSEE, Time Use survey 2009-2010 — 64 to 72% of domestic hours done by women depending on the scope. The gap is narrowing mostly because women do less, not because men do more. ↩

6. The project's public Matrix room (log cited in the chapter "The closed circle"), March 16, 2022: "such users should stick to Adobe products please". ↩

7. Veblen, T. (1899). *The Theory of the Leisure Class*. New York: Macmillan https://www.gutenberg.org/ebooks/833 — leisure time as proof of status, and the displayed contempt for productive work as a caste marker. Cautious use: conceptual echo, not evidence about free software. ↩

8. Cowan, R. S. (1983). *More Work for Mother: The Ironies of Household Technology from the Open Hearth to the Microwave*. New York: Basic Books https://search.worldcat.org/title/9619681 — domestic technologies, presented as liberating, redistributed and increased women's work; designed outside the experience of those who did it. ↩

9. Akrich, M. (1995). "User Representations: Practices, Methods and Sociology", in Rip, A., Misa, T. & Schot, J. (eds.), *Managing Technology in Society*. London: Pinter https://shs.hal.science/halshs-00081749 — origin of the *I-methodology* concept: the designer takes himself, without reflexivity, for the representative of the user. ↩

10. Oudshoorn, N., Rommes, E. & Stienstra, M. (2004). "Configuring the User as Everybody: Gender and Design Cultures in Information and Communication Technologies". *Science, Technology, & Human Values*, 29(1) https://doi.org/10.1177/0162243903259190, 30-63. The two projects the article studies both fail — it stands as an *a contrario* demonstration: without counter-forces, I-methodology wins. ↩

11. Thread "Another dt UI discussion", https://discuss.pixls.us/t/58799, opened 2026-06-22, closed 2026-06-26 after 246 messages. Messages 228, 230 and 233 (2026-06-26), verbatim: "So, let me voice it: I do like a lot the way the UI is"; "Darktable devs are also photographers. And I suppose we can say: The Darktable devs like the way the UI is"; "No you can not. The project has chosen for good reasons: 'Darktable is created for photographers, by photographers'". The community doxa, older (t/27818 https://discuss.pixls.us/t/27818, 2021): "It's made because the developers wanted to make a piece of software that suits them." And the premise was stated backstage as early as 2023 — project Matrix room, January 19, 2023, the documentation maintainer: "I'd assume most (all) devs are users so they would consider themselves to be a key audience" — I-methodology stated as self-evident, 3 years before the public confession. The backstage says something else than the stage: on the project's Matrix room (log cited in the chapter "The closed circle"), the documentation maintainer wrote as early as 2023-2024 — "Something is added that was (IMO) a

bad idea… this made the ui look cluttered, and instead of removing it as a bad idea another change was made to make it less bad" (March 20, 2023); "I can't stomach the endless arguments with [the maintainer], which often end up with worse UX as he tries to satisfy everyone" (March 23, 2023); and, on January 13, 2024, in so many words: "current interface is just bad". The public "we like the UI the way it is" was never the verdict of the backstage. ↩

12. Aphorism by Alberto Brandolini, software engineer, formulated on January 11, 2013: "The bullshit asimmetry [sic]: the amount of energy needed to refute bullshit is an order of magnitude bigger than to produce it". An adage, not a scientific result — but discussed as such in the scientific press: Williamson, P. (2016). "Take the time and effort to correct misinformation". *Nature*, 540, 171 https://doi.org/10.1038/540171a. ↩

13. Thread "Aurelien said: basecurve is bad", https://discuss.pixls.us/t/29055, opened on January 23, 2022 (archived: https://web.archive.org/web/20260723151115/https://discuss.pixls.us/t/29055). Verbatim of the opening message: "people who keep ignoring them often complain that Darktable is bad, scene-referred sucks, and why-the-fuck-did-you-change-display-referred-that-got-me-faster-results". ↩

14. Tronto, J. C. (1993). *Moral Boundaries: A Political Argument for an Ethic of Care*. New York: Routledge https://doi.org/10.4324/9781003070672 — the 4 phases of care, the first of which, *caring about*, is noticing the need;

and "privileged irresponsibility". The origin of the field is Gilligan, C. (1982), *In a Different Voice*, Harvard University Press https://doi.org/10.4159/9780674037618 — cited here for the history of ideas, its empirical results being contested. ↩

15. Russell, A. L. & Vinsel, L. (2016). "Hail the Maintainers". *Aeon*, April 7, 2016 https://aeon.co/essays/innovation-is-overvalued-maintenance-often-matters-more : "The most unappreciated and undervalued forms of technological labour are also the most ordinary… feminist theorists have long argued that obsessions with technological novelty obscures all of the labour, including housework, that women, disproportionately, do." Developed in Vinsel, L. & Russell, A. L. (2020). *The Innovation Delusion*. New York: Currency https://www.penguinrandomhouse.com/books/576816/the-innovation-delusion-by-lee-vinsel-and-andrew-l-russell/ . ↩

16. "Does the world need another incomplete photo editor?", https://discuss.pixls.us/t/975 , March 6-7, 2016 (archived: https://web.archive.org/web/20250907023642/https://discuss.pixls.us/t/975). Verbatims: "People write (and share) things that scratch their own itch" (no. 3); "it's inappropriate to tell them to work on something more useful" (no. 8); "I do Filmulator for me. I give it away because I can, not because I think you should use it" (no. 14); "Freeloaders are a burden and not an asset" and "that's the implicit contract that comes with free software" (no. 10, a Darktable core developer); "most of which are based on ignorance" (no. 21); "how are you offering to help out?" (no. 16); the GPL's no-warranty

clause quoted in reply (no. 28); the final reframing "how could we better utilize developers to more completely meet the needs of typical users?" (no. 27). The thread also contains a real architecture answer (no. 4) and two developers defending the users — the ethos won the thread, it did not reign there without a fight. ↩

17. 3 documented regimes of counter-forces. Users who carry weight before design: the UTOPIA project, where 5 print workers, mandated by their union, designed on mock-ups a typesetting system that started from their trade — the chapter "Thinking against oneself" gives the account and the sources; the pilot system was never commercialized, for lack of capital at the manufacturer — it is the method, participatory design, that survived. Designers who started from a user other than themselves: the OXO Good Grips kitchen line (1989-1990), drawn for the arthritic hands of Betsey Farber and become a mass-market success — a textbook case of universal design, entered into MoMA's collection (Smart Design, *Good Grips*, 1989) https://www.moma.org/collection/works/3758 ; and the "curb-cut effect": the curb cut wrested from Berkeley by disabled activists (first official curb cut in 1972) serves strollers today, deliverers and suitcases — designing for the case the room does not see improves the object for everyone (Blackwell, A. G. (2017). "The Curb-Cut Effect". *Stanford Social Innovation Review*, 15(1) https://ssir.org/articles/entry/the_curb_cut_effect). Finally, users who design for themselves: von Hippel documents entire fields where innovation comes from users — 22% of

the surgeons in a sample of university clinics modifying their instruments, kitesurfing equipment invented by practitioners then taken up by the manufacturers (von Hippel, E. (2005). *Democratizing Innovation.* Cambridge, MA: MIT Press https://doi.org/10.7551/mitpress/2333.001.0001 , chap. 2 and 7). ↩

26. Three-tiered violence

A user stumbles on a maneuver, tries again, fails, and concludes: "I don't understand any of this, I'm no good with computers." The sentence seems harmless, an ordinary modesty. It is the finished product of a violence in 3 stages. First stage, so old that nobody sees it anymore: he was *forced* to go through a computer for a task he used to accomplish perfectly well without one. Second stage: a tool was designed that he could not use — a design failure, in the sense of the preceding chapters. Third stage, the most efficient: he was brought to carry the blame for it. The failure he charges to his own inability has two causes, though, and neither of them is him: nearest at hand, a badly designed tool; upstream, the obligation to go through that tool, which he had never asked for. The first violence enlists, the second excludes, the third has the excluded sign the IOU. It is the third that locks the whole, because an excluded person who believes himself responsible for his exclusion does not protest — he apologizes. Naming these 3 stages means giving anyone who suffers them, or inflicts them without seeing it, the means to break the chain.

The first tier is conscription, and nobody sees it anymore: almost no one *chose* computing. It arrived as a condition of access to things people already did — filing taxes, enrolling children in school, writing to an administration, developing photographs. These tasks got done, and often well, with paper, a counter, a

photo lab; within a generation, the manual route was closed — not outcompeted: *removed* — and the computer became an obligatory passage. When the only way to pay your taxes goes through the Internet, the computer stops being an option: it becomes a required skill and a hidden cost — a tax on access to your own rights. The chapter "The instrument and the machine" described what this forced conversion did to artists: it drove them out of the very medium they worked — paper, negative, matter, replaced by data that only engineers know how to touch. The same gesture was imposed on everyone, and it was dressed in the one word that forbids debating it: *progress*. You can't stop progress, can you? Forcing people who do not understand computers to go through them for tasks they had mastered is a form of oppression; disguising it as history's inevitable march is a form of psychological violence, for it turns a decision — made by someone, somewhere, for debatable reasons of cost and control — into a natural fatality, against which protesting seems as absurd as protesting against the rain. Nobody voted for the closing of the counters. People simply found, one morning, that they were closed.

The second tier is already known to this book: it is the technical failure of the conscripts least armed for the digital battle, measured in the chapter "Counting the absent" (half the world cannot install software) and diagnosed in the chapter "Freedom in a void" (the freedom of a right you lack the means to exercise). Research on the digital divide has shown that the real gap is no longer access but *skill*: once everyone is connected, differences in ability remain, and they stratify by education level[1]. "It's free,

just use it" masks that gap instead of closing it, exactly as the chapter "Freedom in a void" showed for the word *freedom*. And Donald Norman established the design side, cited in the chapter "There is no technical problem": what we call "human error" is almost always a design error, a fault the product throws back onto the user, for want of having been designed for him[2]. The second tier, then, has nothing fated about it: it is bad design, repairable by good.

The third tier is the moral operation, and it has an old name. William Ryan called it *blaming the victim*: faced with a social failure, you study the victim, find a deficiency in her, and forget the structure that produced it[3]. Ryan was aiming at well-meaning progressives, convinced they were helping — and the aim is exact, for the free-software milieu is well-meaning, convinced it is offering freedom, and yet practices the operation without being aware of it. Its everyday form is the culture of RTFM — "read the fucking manual" —, the presumption of user error, the "you didn't search hard enough" thrown at whoever reports an obstacle. The distinctive trait of this form is not harshness — it is that it holds even when the report turns out to be *right*. A user finds the default theme unreadable; a core developer answers him that it is "a design decision for the best of our user's experience" and invites him, if he is not happy, to stop posting, "or maybe you are just a troll!" — and the next version of the software ships a higher-contrast theme, proving the presumed troll right. Another reports a styling defect; user error is presumed and the answer comes: "Then use something else. We don't need this kind of attitude here" — the defect was real, and it

was fixed[4]. The blame precedes the examination and survives it: the sign that the point is not to find out who is right, but to silence whoever disturbs. Norman, again, named the result on the user's side: from having the product's defects imputed to him, the user internalizes them, and Norman speaks explicitly of *taught helplessness* — the taught and inflicted counterpart of the psychologists' learned helplessness[5]. The word is right: the inability is not innate, it is taught.

How does such a teaching hold? Through what Pierre Bourdieu named symbolic violence: the power to impose meanings as legitimate while concealing the power relations that found them[6]. The violence is not the exclusion; it is obtaining that the excluded holds his exclusion to be legitimate. Bourdieu and Passeron had described its model at school: working-class students read their low statistical chances as a personal inability, and *eliminate themselves* — self-elimination, which spares the system the trouble of excluding, since the victim takes care of it[7]. I make the hypothesis — no study follows the same individual from school to software — that the sentence "I'm no good with computers" is the same gesture, prolonged. School taught part of the population that it was not made for technical abstraction; software harvests that ready-made conviction and confirms it, with every unreadable manual, every error message in engineer's language. Psychology supplies the fine mechanics of the internalization: internal, stable, global attribution — not "this maneuver failed" but "I am incapable, with all computers, forever"[8] — and the self-efficacy spiral, where low confidence breeds avoidance, which breeds confirmed incompetence[9].

Here the burden of proof must be reversed. Common intuition treats passivity before the machine as a personal defect: the timid user supposedly lacks curiosity or daring, and should shake himself up. Fifty years of research on learned helplessness say otherwise: passivity is nobody's defect, it is everybody's starting point. What is learned is control — the discovery that one's own actions have effects — and that learning only happens if the environment responds predictably to attempts[10]. Daring before a tool is therefore not an entry quality: it is a product of the tool. A piece of software that never yields legibly to a beginner's gestures, that punishes exploration with data loss or a cryptic error, never lets control be learned. "I don't touch anything, I'm afraid of breaking everything" is thus not a character trait: it is the exact, and predictable, response to a tool that has never rewarded an attempt. We meet the safety of the chapter "Measuring without lying to yourself" from the other end: the hover that destroys data is not merely dangerous, it teaches helplessness.

The objection is that of personal responsibility, and it has its dignity: not everyone will make the effort to learn, some do not want to, and you cannot hold a tool responsible for the laziness of someone who refuses to get down to it. Its core must be conceded: skill has a cost, learning is real, and the chapter "In praise of the simple" defended legitimate difficulty — the trade's, which the tool has no business removing. But the objection confuses two things this book has kept apart from the start: the difficulty of the trade and the difficulty of the tool. Learning

photography is work; deciphering an error message written for an engineer is not — it is a tax.

The border between the two, besides, is not a matter of opinion: it has a criterion, and the criterion is historical. Photography existed before the computer. Exposing, controlling the contrast, holding back one zone and burning in another — dodging and burning —, correcting a color cast, choosing your grain: all of that was already learned in the darkroom, and that difficulty — the trade — cannot be reduced without amputating the capacity of the one who works. Nothing, by contrast, under the enlarger required understanding a file format, the order of a pipeline, a display profile or the jargon of an error message: that difficulty was born with the tool, and it can be reduced without taking anything from anyone. If the distinction has become so hard to see, it is because the digital swallowed the trade whole. For whoever began photography with software — the majority of practitioners, by now — the tool and the trade arrive welded together: every difficulty presents itself as a software difficulty, and one no longer knows what one is entitled to demand of the tool nor what one still owes the trade. Only those who practiced before computing can measure the border from memory: they know what belonged to photography before it entered the computer, and what the tools added to it on their own initiative. This book's criterion is theirs: whatever already existed under the enlarger belongs to the trade; the rest belongs to the tool, and has to justify itself. And above all, the objection mistakes the filter for nature. The cockpit of the chapter "Listening without obeying" already refuted it: the "average" aviator did not exist, and it is the

adjustable seat, not the selection of pilots, that solved the problem. The demonstration also exists for computer skill itself: a major American computer-science school, faced with 7% women in its cohorts, brought that figure to 42% in 5 years — without lowering the bar, by changing the culture of reception and the implicit prerequisites[11]. 7% was not the measure of a natural aptitude of women for computing; it was the measure of a filter, and the filter proved dismountable. What was taken for the map of talent was the map of the welcome.

I have, moreover, the proof by my own practice. For 3 years, I gave private lessons on my software — adult students, often with degrees, who had read the documentation and were paying to learn: the most motivated, best-equipped population one could dream of. And many, despite all that, could not get the workflow started. Faced with that finding, only two conclusions are possible: either most educated, willing people are imbeciles, or the design is bad. The second is the only serious one. When the most favored slice of your users stumbles, it is no longer the user who is in question — it is the demonstration, made in optimal conditions, that the second tier of the violence is indeed a design defect, and not a deficiency in people.

Three-tiered violence is thus the exact opposite of everything this book has defended. To design is to solve a human's problem, and to take charge of the translation between his need and the space of solutions; three-tiered violence does the reverse — it botches the translation, then bills the failure to the human. And the right word must be dared for what the milieu inflicts here: the one who most needs to be cared for is treated there as the one who

deserves it least. The conclusion will have nothing to add to this finding; it will only draw from it the deontology that is missing. This violence has a consequence not yet named: it is comfortable for the one who exercises it. As long as the inability lies on the user's side, there is nothing to repair in the tool, nothing to question in the room of the chapter "What insularity produces". Blaming the victim is not merely an injustice; it is the mechanism by which an insular circle excuses itself from reforming — and that is why it does not stop at the users. Turned inward, against the one who, from within, contests, it becomes the engine of a loop that closes on itself. That is the subject of the next chapter.

---

1. Hargittai, E. (2002). "Second-Level Digital Divide: Differences in People's Online Skills". *First Monday*, 7(4) https://doi.org/10.5210/fm.v7i4.942 ; van Deursen, A. & van Dijk, J. (2011). "Internet skills and the digital divide". *New Media & Society*, 13(6), 893-911 https://doi.org/10.1177/1461444810386774 — the divide has shifted from access to skill, and stratifies by education. ↩

2. Norman, D. A. (2013). *The Design of Everyday Things*, revised ed. New York: Basic Books, chap. 2 https://jnd.org/books/the-design-of-everyday-things-revised-and-expanded-edition/ — "human error" is most often a design error. ↩

3. Ryan, W. (1971). *Blaming the Victim*. New York: Pantheon https://search.worldcat.org/title/110095 — study the victim

to find the defect in her, and ignore the structure; the author explicitly aimed at well-meaning reformers. ↩

4. Thread “v4.4 appears grayed-out”, https://discuss.pixls.us/t/40694 : to a user deploring the lack of attention paid to user experience, a core developer of the project replies (2024-10-20) “don’t bother to post here, or maybe you are just a troll!” (archived: https://web.archive.org/web/20260714060151/https://discuss.pixls.us/t/40694); the theme’s contrast would be raised in a later major release. Thread “Problems with setting up a style in 4.8”, https://discuss.pixls.us/t/44459/40 : to a user finding the software unusable, moderation replies (2025-10-06) “Then use something else. We don’t need this kind of attitude here” (archived: https://web.archive.org/web/20260723093928/https://discuss.pixls.us/t/44459/40), then archives the thread. Two cases, cited for the role of those involved; verbatims checked against the snapshots. ↩

5. Norman, *op. cit.* — Norman takes up *learned helplessness* and coins *taught helplessness*: the user, from having the design’s defects imputed to him, learns to believe himself incapable. ↩

6. Bourdieu, P. & Passeron, J.-C. (1970). *La Reproduction : éléments pour une théorie du système d’enseignement.* Paris: Minuit https://www.leseditionsdeminuit.fr/livre-La_Reproduction-1952-1-1-0-1.html — symbolic violence: « tout pouvoir qui parvient à imposer des significations et à les imposer comme légitimes en dissimulant les rapports de

force qui sont au fondement de sa force » (our translation in the body). ↩

7. Bourdieu, P. & Passeron, J.-C. (1964). *Les Héritiers : les étudiants et la culture*. Paris: Minuit https://www.leseditionsdeminuit.fr/livre-Les_H%C3%A9ritiers-1950-1-1-0-1.html — *self-elimination*: the dominated convert a low objective probability into a subjective inability and give up of their own accord. ↩

8. Abramson, L. Y., Seligman, M. E. P. & Teasdale, J. D. (1978). "Learned Helplessness in Humans: Critique and Reformulation". *Journal of Abnormal Psychology*, 87(1), 49-74 https://doi.org/10.1037/0021-843x.87.1.49 — the internal, stable and global attributional style. ↩

9. Bandura, A. (1977). "Self-efficacy: Toward a Unifying Theory of Behavioral Change". *Psychological Review*, 84(2), 191-215 https://doi.org/10.1037/0033-295X.84.2.191 ; applied to computing: Compeau, D. R. & Higgins, C. A. (1995). "Computer Self-Efficacy". *MIS Quarterly*, 19(2), 189-211 https://doi.org/10.2307/249688 — low self-efficacy → avoidance → confirmed incompetence. ↩

10. Maier, S. F. & Seligman, M. E. P. (2016). "Learned Helplessness at Fifty: Insights from Neuroscience". *Psychological Review*, 123(4), 349-367 https://doi.org/10.1037/rev0000033 — the reversal at 50 years: passivity is the default response; it is control that is learned, provided the environment responds to actions. ↩

11. Margolis, J. & Fisher, A. (2002). *Unlocking the Clubhouse: Women in Computing*. Cambridge, MA: MIT Press https://mitpress.mit.edu/9780262632690/unlocking-the-clubhouse/ — Carnegie Mellon brought the share of women in its computer-science admissions from 7% to 42% in 5 years through a reform of the culture and the implicit prerequisites, without changing the academic requirements. ↩

27. The closed circle

The two preceding chapters looked at the room and its violence toward the outside. This one looks at what the same machine does on the inside — to its own members, and first to the one who contests from within. Its thesis fits in a definition: a closed circle is not a group that agrees; it is a group whose immune system attacks disturbing information, and whoever brings it. Disagreement there is not refuted, it is expelled; and the expulsion is then rewritten so that the circle has nothing to learn from it. This loop can be read twice: first in the literature, which has measured it on the big projects; then in the case I know from the inside, verifiable exhibit by exhibit since all of it is public and archived.

The general loop first, as correlation and mechanism, not as simple causality. First arc: aggressiveness is endemic and normalized in the high places of free software — on the Linux kernel mailing list, the milieu's cultural reference, the analysis of 1,500 code-review emails found incivility installed as the ordinary register[1]. The style is that of ritual combat: an idea is not discussed, it is attacked, and holding up under attack is part of the game — a culture of argument-as-confrontation that linguistics has described well beyond software[2].

But this aggressiveness, already studied because it is easy to see, also has its invisible counterpart: when it comes to systemic

violence, aggressiveness is the kind you notice immediately, the kind that is easy to name. And the whole free-software ecosystem should not be reduced to the single case of the Linux kernel. There exists a duller, more devious, more hypocritical violence, which never says a word louder than the next and stays courteous at least in form. The project I left practiced little invective — it wore people down: reports left to rot until the robotic closure, criticisms dismissed without examination, complainants sent back to their supposed incompetence. A polite toxicity, almost invisible when you skim the threads judging only the form. We must pause here a moment on what violence means. Daisy Letourneur gives a general definition: violence is removing options from people[3]. I narrow the spectrum: it is removing options, arbitrarily, without demonstrated necessity. If you accept that "one person's freedom ends where another's begins," violence is any restriction of someone's freedom well before it starts encroaching on anyone else's. Thus, preventing people from polluting the environment shared by all is not violence against the polluters; it is simply protecting everyone's right to live in an environment that does not make them sick.

My case study practiced almost no open aggression — the only core member with a truly abrasive register had left the core years earlier, to maintain nothing but a third-party library. On the forum as on the repository, people are polite, and that is almost worse. Violence there takes the twin form, which research knows just as well. Low-intensity incivility, first, defined precisely by the ambiguity of its intent — deniable enough to remain unsanctionable[4]. Ostracism, next — ignoring, sidelining, leaving

without an answer — which psychology has shown activates the same brain regions as physical pain and strikes 4 fundamental needs: belonging, self-esteem, control, the sense of one's own existence[5]. Then symbolic violence, whose workings on the general public the chapter "Three-tiered violence" described: imposing a reading of the situation that makes the one who suffers the defect carry it. Turned inward, it takes on an additional function, and that is the one this chapter is after. When a user reports a breakdown and is told that he is going about it wrong, or that he is not the intended audience, he is not merely being put down: a piece of information is converted into a personal defect. The report came in as a fact about the software; it goes out as a fact about the user. Nothing has been learned, and the circle has won the right to change nothing.

Above that still, epistemic violence, which no longer consists in blaming the user for his experience but in stripping him of the right to tell it. The chapter "Listening without obeying" had delimited the only domain where the user has undivided authority — his goals, his context, his pains: what he has lived through, he alone disposes of. Rewriting it in his place — "it's not a bug, it's a design choice," "nobody else complains," "you didn't understand what you saw," "satisfied users are less noisy" — takes from him the only competence this book grants him entirely. Christiane Rochefort gave it the formula I have not managed to improve: the oppressor does not hear what the oppressed says as a language, but as a noise[6]. Requalifying speech as noise — background noise, complaint, mood, incompetence — is the most economical operation there is, since it dispenses with answering

and does not oblige you to contradict it. Philosophy has named it: assigning the speaker a credibility deficit that dispenses with evaluating what he says is an epistemic injustice[7]. This precise point has been measured. The qualitative analysis of 203 bug-report threads on GitHub, 81 of them toxic, shows that toxicity there arises first from diverging perceptions of the defect's severity and priority — that is, at the exact spot where it is decided who has authority to say what matters — and that these exchanges reduce the likelihood of a concrete follow-up, such as linking the report to a pull request[8]. The report is not merely badly received: it stops producing effects. And at the scale of entire projects, the prevalence of toxicity varies inversely with the issue-resolution rate — an association measured on 2,828 repositories, not an established causality[9]. We will see further on its extreme form, applied no longer to the user, but to the contributor who contests.

Finally, the technique of inexhaustible civil questioning, widespread enough online to have received a name, *sealioning*: politely, indefinitely demanding proofs and justifications — often on elementary or already documented points — until the target's patience runs out and, by snapping, the target alone puts himself in the wrong[10]. That is the economy of polite toxicity: the one who wears down is never sanctionable, his violence is passive, invisible; the one who gets worn down ends up being sanctionable, especially if he snaps, which justifies after the fact the way the group treated him. The dissident is not beaten there, he is turned away — a thousand times, courteously, until he walks out. And the sorting this regime performs has a direction,

which the chapter "What insularity produces" documented on the stripes: difference is punished there, exigence passes for elitism, adherence to scientific rigor for a lack of artistic sensitivity. There is a word for an order that rewards conformity to the mid-range and evacuates both excellence and deviation — mediocracy: "the mediocre have taken power," and its emblematic injunction could serve as the regime's motto: "put away those complicated works, the account books will do"[11]. An organized mediocrity has no need to attack: dismissing is enough.

And the dismissal has its instrument of choice, which must be dismantled here because this chapter's case will show its official application: tone policing. Treating a speaker's anger as a defect of form dispenses, precisely, with examining his reasons — Fricker's credibility deficit, applied to angry testimony[12]. The procedure has a history, and it is on record: it is the one that filed women's anger under the diagnosis of hysteria — pathologizing the emotion so as not to have to hear what it was saying[13]. Audre Lorde dismantled it in one sentence, as early as 1981, reporting the injunction addressed to her — "Tell me how you feel but don't say it too harshly or I cannot hear you" — and turning the question around: is it my manner that keeps her from hearing, or the threat of a message that her life may change[14]? Anger is loaded with information, she said. A collective that judges it instead of reading it deprives itself of exactly the information that the social sensitivity of the chapter "The committee will invent nothing" serves to capture, and irreproachable form becomes a tribunal there: what one refuses to settle on the merits gets settled on tone — I stayed courteous,

you did not, therefore I am right. That is why false politeness is not the opposite of rudeness: it is a refusal to hear, in evening dress. Reading the other's state in order to hear him is the gesture with which everything that deserves the name of care begins; reading it in order to use it against him is the exact gesture with which it ends.

Second arc: this norm — openness or politeness — is rationalized by ideology. The rhetoric of openness requalifies the complaint as censorship and protects the "difficult personalities." But the expression itself is a trap, and it must be dismantled rather than taken up, for it judges a behavior outside its history and its context. A behavior is a result: on a snapshot, without the chain of interactions that leads to it, defense looks like aggression and resistance like provocation. The psychology of attribution has described the bias — the observer imputes to character what the actor owes to the situation — and the meta-analysis forces us to state it exactly: the asymmetry, near zero in general, subsists precisely for negative events, that is, where blame gets handed out[15]. The chapter "The well-tempered conflict" will name conflict debt; the one who finally blows up after years of polite attrition supplies the snapshot with exactly the expected image — and the attrition, for its part, appears in no photograph. There is worse. In the polite regime of the first arc, the label does not only protect real aggressors: it manufactures them. The sociology of deviance established it 60 years ago — "deviance is not a quality of the act the person commits, but rather a consequence of the application by others of rules and sanctions"; "the deviant is one to whom that label has successfully been applied"[16]. So a

criterion is needed, and it must be usable by someone who has no prior opinion of the people involved — failing which everyone will file the others in the box chosen in advance. This criterion fits in two questions, bearing not on people but on what they say: what does the statement aim at, and by what yardstick does it measure what it says?

The first question separates the person from the work. Attacking someone for what he is, humiliating him, opening hostilities with no history: nothing redeems these conducts, not even being technically right. This division is not my invention: work psychology operationalized it nearly 40 years ago under the name of *destructive criticism* — criticism that is non-specific, inconsiderate in its form, and that imputes poor performance to internal causes, that is, to what the person is rather than to what she did[17]. The third arc will cite its measured effects in code review; what serves here is its definition, for it gives exactly the line we are looking for. There are people a community must exclude — the racist, the sexist, the one who attacks without prior provocation —, and putting them out is not labeling: it is a legitimate policing, which a community cannot refuse without abandoning its members.

The second question separates two criticisms that are constantly confused, and it is the one that was missing. A criticism aimed at a decision is measured by one or the other of two yardsticks. Either it takes the project at its word — you announce this, your records show that: it then imports no foreign requirement, it demands an account of commitments already made, and the project cannot brush it aside without going back on its word. Or

it measures the project against a yardstick the project never adopted — "make it a free Lightroom": it is then a request, not a failing. Receivable as a request, and just as refusable as a request, for 12 volunteers working between 5 and 10 p.m. will not catch up with a multinational; saying so is not closing the circle, it is holding a perimeter — an exercise the chapter "Stating the problem" made into a discipline.

From there, 3 cases, and 3 treatments. The attack on persons excludes itself. The out-of-perimeter request gets refused, arguing the perimeter and not the person who carries it. The immanent critique gets examined — and it alone, refused, closes the circle. The merit of this division is not its subtlety: it is that it bears on statements and not on individuals. The same person will produce all three depending on the day, and it is by judging people rather than their sentences that the boxes get manufactured.

This division nevertheless has a limit, which must be stated as plainly as the rule — failing which the rule becomes an immunity. Judging statements rather than persons holds at the scale of an exchange; it stops holding at the scale of a trajectory. An isolated act says nothing about the one who committed it; a series of acts sustained over years, with no error ever acknowledged and no lesson ever drawn, ends up saying something — and that something is a finding, not an insult. Past that point, refusing to name the person no longer protects fairness: it protects the individual against the balance sheet of his own acts, and deprives the collective of the only information that would be useful to it. The chapter "The incompetent worker" will give this balance sheet its rigorous form: competence is judged on a balance of

actions over time, not on a one-off gesture isolated from its history. Naming someone at the end of a documented series is therefore not attacking him, it is summarizing exhibits already filed. The line is double: on an exchange, one judges sentences; on a decade, one may judge a record — provided it is made of dated exhibits, and ones that can be put to the person they concern.

The fact remains that this division, in practice, is almost never applied symmetrically — and that is where the scandal lies, not in the categories. The same conduct changes name according to the position of the one who holds it. The milieu supplies the best-documented case, and it is its cultural reference: on the Linux kernel list, the incivility measured in the first arc was held, for years, to be one man's style rather than a problem — until the man himself publicly apologized for it in 2018, withdrew for a while, and the project replaced its "code of conflict" with a code of conduct[18]. What had covered this conduct for so long was not a doctrine of civility: it was the position of the one who held it. The written remedy does not suffice on its own: the first empirical study of free-software codes of conduct notes that their application remains uneven[19]. Symmetrically, the contester receives the label at the first misstep, and often without one. In my case study, "difficult personality" is thus the name given to whoever deviates from the milieu's ethos and habitus — the demanding one, the professional, the theoretician: all "difficult," because they force the milieu to question itself. And the treatment applied to the labeled — the turning-away, the attrition, the *sealioning* — ends up producing in some of them the conduct that justifies the label retroactively when they snap

loudly, rather than withdraw in silence. Here again, the measure exists: recipients of destructive criticism report handling subsequent disagreements by resistance or avoidance, no longer by collaboration or compromise[17]. The prophecy fulfills itself, and the circle holds its proof.

This asymmetry of application is not an accident of mores; it is an effect of position, and it must be stated in general terms, for it will be found again at every step of the case. The trap here is that the violence of the *establishment* (which I call right-wing, conservative violence) can express itself without expression: simply by maintaining the status quo, in institutions where it is denied that they are unjust and exclusionary for some, and where even exploring the hypothesis as a mere thought experiment is refused. Winning a fight without fighting, simply by locking the decision protocols, having nothing else to do but wait for the opposition to tire and then exhaust itself, is surely the most odious of privileges. Conversely, the violence of the oppressed (which I call left-wing, progressive violence) is an act of resistance that uses the means available — noisy, disorganized and sometimes clumsy, but above all: visible[6], and that is their whole offense. The intellectual trap that awaits us here would be to believe that, because these two violences oppose each other, they are symmetrical: they are not — neither in means, nor in the forces involved. The moral trap would be to condemn the visible violence for its form, ignoring what it opposes: an inert colossus with not the slightest desire to move. Contemporary history is good at calling “resisters” those people whom Vichy France and the German occupier called “terrorists,” for they assassinated,

sabotaged and carried weapons in peacetime (the armistice having been signed on June 22, 1940, France was administratively at peace). There is thus nothing more bourgeois than condemning "all forms of violence" for their mere form, without looking either at their context or at the other realistic possibilities of struggle offered to their authors, for we know perfectly well how to requalify violence after the fact, to make it tolerable when it suits us, just as well as how to hand out medals to all those "terrorists."

It remains to say why the balance sheet of trajectories is never drawn, and the answer lies in a doctrine, stated publicly. In June 2019, at the annual meeting of free graphics software, the project's two principal craftsmen gave a talk for its 10th anniversary, devoted in part to the "different approaches of team organisation"; that is where one slide sums up the project's constitution in one word: *do-ocracy*. The same slide gives its single article, and it must be read slowly: "if there is a rule, it's don't stop anyone from coding when they're having fun"[20]. The word is not a quip. It designates a catalogued model of governance, in which those who do are those who decide, and in which authority is measured by the time and energy invested[21].

One must see what this formula settles, and what it forbids. It settles legitimacy without election or mandate: to do is to be entitled to decide. It forbids, by the same stroke, any evaluation of what is done — for if the title to decide proceeds from the act, the act can no longer be judged without the title collapsing. Criticizing the work of the one who does then amounts to contesting his right to be there. Do-ocracy does not distinguish

quantity from quality, and cannot: it has no other unit of account than the fact of having acted. Hence the answer opposed to every criticism — "write code," "do it yourself" — which is not rudeness, but the exact application of the doctrine.

The article, for its part, says even more than the word, and its form deserves a pause. The only rule the project gives itself is a *prohibition on refusing*. It sets no acceptance criterion, no quality threshold, no maintenance obligation: it bounds authority in a single direction, that of the no. Its criterion, next, is the fun of the one who codes — not the user's need, not the software's health, not the maintenance burden left to those who will stay, but the contributor's pleasure, posited as what the organization must protect first. The rest follows from it with an almost mechanical regularity. The chapter "Non-heroic ambition" measured a refusal rate of pull requests that collapses as the right to merge concentrates: that is the constitution applied — nothing gets refused, since the only written rule says not to stop anyone. And the chapter "What insularity produces" collected, 4 years later, what becomes of this rule the day someone finally proposes acceptance criteria: they are brushed aside, among other grounds, as "impacting developer fun." The 2019 slide and the 2023 testimony say the same thing 4 years apart, one from the rostrum, the other from the forum: fun is the criterion, and it prevails.

That is the origin of the rest. A balance sheet supposes a yardstick external to the act, and the doctrine admits none. The ordinary instruments of accountability — the election, the jury, the term-limited mandate — are nowhere to be found in these informal

governances[22], and the chapter "The incompetent worker" said what remains when they are missing: not indulgence, but ostracism, which sidelines without ever having to state grounds. Do-ocracy therefore does not produce a community without power: it produces a community where power is not discussed, because it merges with the work. The chapter "Non-heroic ambition" measured its sharpest consequence: the right to refuse pull requests dies out as it concentrates.

The project must immediately be given its due, for the next slide complicates the picture — and the complication is more instructive than the grievance. It is titled "responsibility," and it states: "whether we like it or not: people depend on our software to get their work done"[23]. The sentence is right, and few free projects state it so plainly. There is thus no denial: the users' dependence is seen, named, assumed.

It remains to see into what this responsibility is converted. The slide gives it a single content: the promise that old edits will remain readable indefinitely — in 10 years you will find your photograph's sidecar file, and the software will reproduce its result exactly. The commitment is real, technically costly, and the project has held to it. But observe the register. On everything else a user depends on to work — understanding the interface, getting an answer to a report, not seeing his know-how canceled at every release —, nothing. The responsibility is acknowledged, then discharged in the only register where it can be discharged by writing code. The responsibility toward the user bears only on the technical artifacts the software has produced.

This is the gesture the chapter "There is no technical problem" has been describing since the beginning of this book, applied this time to a duty and not to a problem: an obligation toward persons receives a technical reading, and becomes a property of files. The choice is not arbitrary — it is the one that leaves the constitution intact. Keeping a format promise requires consulting no one, evaluating no work, refusing no contribution: it is done by coding, hence without ever stopping anyone from coding. The responsibility thus defined costs no governance. The other half — the one that would oblige to arbitrate, to refuse and to maintain — remains without a holder.

The talk does not stop there, however, and the next slide is the most important of the three. It is titled "lessons learned," and it puts to the balance sheet the question of the model: what was gained from *laissez-faire* — the word is theirs? Four answers. The first 3 are real gains: quick development, features the speaker would never have thought of, translations into languages he never knew existed. The fourth fits in 2 words and an emoticon: "technical debt :/"[24].

One must measure what this line establishes, for it spares this book a good part of its work. The link between the absence of refusal and technical debt is not a thesis I plead against the project: it is a finding the project drew up itself, in public, before its peers, and entered at the 4th rank of its own balance sheet in 2019, in the talk that celebrated (a little early) its 10 years of existence. There is nothing to impute: the diagnosis is signed.

What remains is what followed, and this is where the case becomes probative. A diagnosis commits to nothing as long as it does not modify the rule that produced it — and this rule was the constitution's only article. Amending it would have meant stopping, sometimes, someone from coding. The chapter "Non-heroic ambition" gives the sequel in figures: the refusal rate of pull requests, which stood at 19% the year of this talk, falls to 10.9% in 2021, then to 8.5% in 2023. After the finding, laissez-faire did not recede: it deepened. The lesson was learned in the strict sense of the word — stated, written, presented before the assembly of free graphics software — and never applied.

We recognize here a figure this book has already crossed twice. The chapter "The cargo cult" describes practices whose form is reproduced after their reason has been lost; the chapter "Stating the problem" follows the agile method along that path, until its own signatories disown it. Laissez-faire is the governmental version of the same movement. The rule really worked — in the conditions of its beginnings: little code, few users, no legacy to hold, and error without consequence. Those conditions have disappeared; the rule has remained. What Lehman's law, cited in the chapter "Non-heroic ambition", states of software — it changes ceaselessly, so what holds it up must be reworked — holds of its governance too: a constitution written for a few people having fun does not govern a tool that professionals depend on.

The comparison has a limit, though, and it plays against the case rather than in its favor. The cargo cult, in Feynman, supposes ignorance: the gesture is repeated without knowing what gave it

its efficacy. Here, no one was ignorant of anything — the 4th line of the balance sheet named the cost, and the 3rd slide named the users' dependence. This is thus not a rite maintained for want of having understood, but a rite maintained although it had been understood. The chapter "Thinking against oneself" will show that this milieu sometimes knows how to look at itself squarely; the case adds here the precision that was missing: looking at yourself serves no purpose as long as nothing obliges you to draw the consequences.

One trait is still missing to understand why everything, in such a milieu, degenerates into combat and must pass through it as soon as you care about the direction the project takes: there exists no tribunal. The "structureless" bodies of the chapter "The committee will invent nothing" do not only have informal elites; they also have no arbitration body — no judge, no appeal procedure, no one empowered to settle a disagreement over the project's direction, a design choice or an implementation, and above all no one to dismiss the repeat-offending incompetent, nor for that matter to measure the incompetence. Yet disagreements do exist, and the chapter "The word and the thing" showed that they do not die out on their own. For lack of an arbiter, each therefore fights with the weapons at his disposal, for what he believes is right: one has the merge rights, and merges; another has the forum's moderation, and closes the threads; a third has free time in abundance, and wins by attrition; the last has only public speech, and all that is left to him is to campaign. It is a Wild West, but by absence of a judicial institution: conflict there is settled by means, never on the merits, and victory goes to the most

available or the best endowed with access rights, independently of the validity or the rationality of the positions. Here we find again the asymmetry of the two violences described above: whoever holds the access rights has nothing to do to prevail, he need only not move; whoever has only speech must act, hence show himself, hence expose himself to the reproach of form. The chapter "Decide, then hold" gave decisions a closure rule; here, the only closure rule is the exhaustion of one of the camps. Each of these weapons will be recognized in the case that closes this chapter.

Third arc: the costs of the norm are not evenly distributed. The study of destructive code reviews shows that they demotivate everyone, but that they push women and non-binary people toward the exit in a markedly higher proportion[25]; and the analysis of millions of contributions shows that women's contributions are better accepted than men's — as long as their gender is not identifiable, acceptance dropping below men's as soon as it is[26]. This is also the answer to the objection of candor — the milieu is simply direct, and the touchy just need to toughen up: a "candor" whose measured costs fall asymmetrically on certain groups is not a communication style, it is a population filter.

Fourth arc, which closes the loop: those who leave look precisely like those who were already missing (chapter "What insularity produces"), the sorting strengthens, and the norm that drove people away finds itself confirmed as "the culture of the milieu." A norm born in insularity, rationalized by its values, whose costs

expel whoever does not resemble it: the circle closes by itself, without any member needing to want it.

The picture must immediately be completed, for the circle's violence does not only descend toward the minorities: it also climbs back up. The systematic study of toxic threads on GitHub established that the dominant motif there is not the confrontation between developers, but **the user in a position of demand against the maintainer**[27] — the siege described in the chapter "The invisible worker", measured. In a closed circle, everyone is by turns shooter and target; it is a regime of interaction, not a quarrel of persons, and whoever enters it ends up playing both roles.

The case shows, dated and archived, each phase of the loop. I present it in the order of the mechanisms, each exhibit being sourced in a note; the reader is invited to verify — that is the whole point of a public case.

None of the mechanisms described here is established by this case: all were described elsewhere, on other terrains, decades before it. What the case contributes is the entire loop, dated, in a single project — and that is why it must be documented: the loop that closed here will close in the same way on any project that runs by the same rules, and it can only be recognized in time if someone has shown it once, evidence in hand. And it must be read for what it is: not the trial of one man by a group, nor of a group by one man, but the functioning of a system — one that silenced many more voices than mine; mine was merely the loudest.

Phase one: the internal voices die. The chapter "Listening without obeying" recounted the "shut up and fix it," and gave its name from the literature: organizational silence. It must first be established that this silence is nothing personal, for it has a history, prior to me. As early as 2016 — six years before my departure —, a user asked on the community's forum that the existing be stabilized before more features were piled on: he was told that users who do not contribute are "a burden and not an asset," that his arguments were "based on ignorance," and the thread was closed the next day — the question of needs, for its part, was never addressed[28]. The forum's record keeps the inventory of what followed: sixteen thread closures for "heating up" from 2016 to 2026[29] — including, in 2021, a user suspended for two months for asking whether the development priorities were the right ones — and a house formula, "nobody is forcing you," attested from 2018 on[30].

Then came the collection-filters affair, and it is worth telling in detail, because in it you see all the voices die together, in three weeks, dates in hand. April 2022: the overhaul of the central photograph-sorting tool — 7,600 lines — is merged at the end of a discussion among 8 participants, and the module it replaces is deleted from the code the same day; the simple solution another contributor had proposed in February 2022 — ten times shorter, and far more elegant in my opinion — had been crushed in passing, and the question of the duplicate, put in writing, never received any answer[31]. Within a few weeks, a dozen tickets oppose the overhaul[32]. The very day of the discovery, in the project's chat room, the documentation maintainer was already

writing that the overhaul "felt like a dev scratching an itch"[33]; the best-furnished ticket is not mine: he is the one who opens it, and it gathers 14 participants and 89 messages — a core developer there judges the new design neither readable nor intuitive, and the community lead asks for the return of the previous interface — and it is still open in 2026. My revert proposal, put to a vote, gathers 6 votes to 4: the maintainer refuses it — "I'm quite opposed to this PR," then, in the neighboring discussion, to the contributor who remains unsatisfied, "Well I cannot force you to be happy :)," and finally the doctrine: "I'm always in favor of moving forward." The grounds opposed to the critics deserve to be collected, for each served against several voices at once: the months of work invested — the sunk cost of the chapter "Decide, then hold", invoked in writing; "most people don't like changing," hatred of GNOME 3 in support; "kids scream from what they want until they have it, not adults" — the reported pain treated with condescension, as childishness; "stay constructive" — what is constructive about silencing the majority opinion to defend a change nobody had asked for? One concession was merged in 77 minutes — the preference restoring the old interface — while the majority revert had died in a week; and the Darktable 4.0 release notes announced the overhaul without mentioning that the fallback preference existed. Three weeks later, the documentation maintainer's report was closed on a cosmetic fix — a tooltip — against the advice of its author, who would reopen it in December. In August, the forum moderator proposed instituting a design review for large changes, citing this affair by name; 7 approvals, closed in 5 days: "Alright. We're already doing the best we can then."[34] In December, a core developer drew up the

assessment from within — "the biggest mistake was to not keep the old one as default" — and the overhaul's principal defender granted that he was right; in 2026, on the same still-open thread, a long-time tester sums up: "Then the originator and me got shut up :-)"[35].

That is why this affair is worth more than a grievance. On a central choice, nearly all the categories of voice — users, testers, contributors, a core developer, the documentation maintainer, the forum moderator himself — asked for the same thing, at the same moment, with a majority vote in support; and none of it weighed against a single man in a hurry to "move forward." I was one of those voices — the loudest. The speech that dies in this phase is not mine: it is everyone's. But I am the one through whom the problem was brought to the users' attention.

Phase two: exit. When voice is dead, said Hirschman, defection remains — and in free software, defection has a technical name: the fork[36]. I forked. And it is here that the chronology, reconstructed date by date, is interesting: the public hostility did not follow my criticisms — it followed my departure. Until early 2022, the community's forum showered me with thanks; the first public attacks concentrate in the very weeks of the fork (May 2022) and of my departure from the project's channels, at the end of June 2022; and my big critical article is 8 months later than this reversal — it arrived in an already hostile field, it did not create it. The very week of my departure, the project's chat room was in fact still approving of me: a regular thanked me "for the work and the rant," another judged the new interface "a bit of a worry"; on the public forum, a third defended my right to say things

bluntly[37]. What the circle punished first was not arguments: it was the defection. The loyalty betrayed, not the thesis defended. The records count the silent defections that followed mine: the chapter "The invisible worker" measured the melting away — commits, pull requests, reviews — of those who left without a word. And the chronology carries a second, finer lesson, for my criticisms did not all have the same register. What coincided with the reversal was a rant video, the *least* technical of my criticisms: ergonomic regressions, said bluntly — and that is the language that was retained, to the point of removing its link from the forum. The most rigorous version came 8 months later: the long-form article, sourced, quantified, examples in hand — built precisely to be hard to counter. It changed nothing, for better or for worse: the sentiment had already turned, and the substance never received an answer on substance. One must weigh what this asymmetry establishes: if the *content* of the criticism had been the variable, the most solid criticism would have produced the strongest reaction — it produced none. The reaction tracked two other variables, form and defection, and neither of the two is an argument[38].

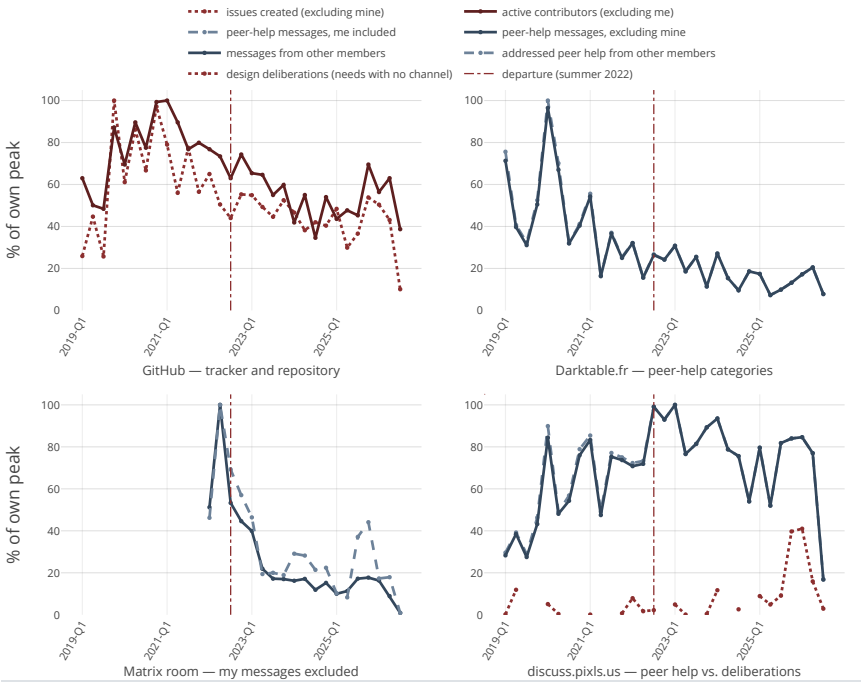
The chronology says when the sentiment turned; the chat room's log — the Matrix bridge of the IRC channel, whose export I have since obtained — says what it was before turning. I was appreciated the way a supplier is appreciated: as long as I delivered new modules and fixed bugs, the esteem was without reserve — the threads of thanks follow the calendar of my deliveries. The only time I spoke, in the project's chat room, of how punishing the load was — that I could no longer stand

repairing what others broke, often things I had already repaired once —, no one offered to relieve me: the tone got noted, and the answer is on file in the chapter "Listening without obeying" in 3 messages — find it, fix it, stop complaining[33]. The affection was indexed on the delivery flow; the person under the flow was nobody's problem. And what this attrition loop — repair, watch it break, repair again — signals technically must be named: a project with no test protocol nor internal validation to contain the regressions. Not for lack of hands — the chapter "Ship, don't dump" counted up to 130 of them a year — but by choice: the guardrails, like the acceptance criteria of the chapter "What insularity produces", would have meant "impacting developer fun." The volunteer worker's comfort ends where other people's begins.

Phase 3: the rewriting. Four years later, the same community tells another story, and the drift can be documented on one and the same speaker. Summer 2022, the forum moderator: "Nobody has been rejected"; "he asked to be removed and I obliged him." Summer 2026, the same man: "it is our fault (particularly mine) for tolerating such poor behavior for so long," while around him others add that the community has "thrived since he left"[39]. "He chose to leave" has become "we endured him for too long": the narrative has migrated from my free choice to their heroic patience, without ever passing through the box "why did he leave?". Social psychology expected nothing else: dissonance gets reduced, and collective memory rebuilds itself in the service of the group's present[40]. Two details give the measure of the reconstruction: in 2022 itself, a user was publicly astonished to be

hearing “the equivalent to good riddance” around the departure of the one who had “brought Darktable to a completely new level”[41] — the climate the official narrative denied was thus audible in real time; and in 2023, a third party described the ongoing discussion as having “all the look and feel of a mob attack”[42] — the term is not merely pictorial: the sociology of work calls *mobbing* this group harassment that converges on a target and ends by excluding it, in 4 phases of which the last is eviction[43]. The circle rewrites all the more freely in that the contemporary witnesses, too, have been archived.

Now the narrative of renewal — the community has “thrived since he left” — asserts a fact, and a fact can be tested. No record confirms it; all show the opposite. The repository’s active contributors peak in 2020-2021, my most active years, then melt away — the chapter “Ship, don’t dump” gave the curves. The project’s Matrix/IRC discussion room, *my messages excluded from the count*, loses half its activity in the 6 months that follow my departure, and four fifths of it in 18 months. The peer help practiced there — the answers addressed by one member to another — falls from 4,900 messages in the year of my presence to 1,600 the following year, and has never come back up[44]. Correlation is not causality, and I do not credit myself with the activity that took place around me during my active years. But “thrived” is an assertion of fact, and no measure supports it: on every metric, the house is emptier than before, and user assistance is in clear decline everywhere, or at best stagnant.


issues created (excluding mine)
peer-help messages, me included
messages from other members
design deliberations (needs with no channel)
active contributors (excluding me)
peer-help messages, excluding mine
addressed peer help from other members
departure (summer 2022)
% of own peak
GitHub — tracker and repository
Darktable.fr — peer-help categories
Matrix room — my messages excluded
discuss.pixls.us — peer help vs. deliberations

The narrative of renewal, tested in 4 places, 2019-2026. *Everything is counted in messages or in events dated to their true quarter; each series is a percentage of the peak of its own reference series; the dashed line marks the departure (summer 2022); 2026-Q3 is partial and excluded from the choice of peaks. The records:*

- ***GitHub****: issues created and active contributors — my reports and my commits are excluded from the counts.*
- ***Darktable.fr****: messages from the peer-help categories only (excluding the mandatory introductions of new members), with and then without my messages.*
- ***Matrix/IRC chat room*** *(open from 2022): total messages, and the reduction to peer-help messages addressed to other members,*

without mine.

- ***discuss.pixls.us**: peer-help messages, under the same conventions, minus the module announcements and previews and the design deliberations; the red dotted line isolates those deliberations — terminology, overhauls, threads of the conflict — which are not peer help but needs recorded without a channel.*

Two readings, developed in the body of the text:

- ***the staggering of the peaks** — the bug tracker and French-language peer help peak in 2019-2020, my most active years; the French-language forum drops off from 2021, the year my participation there collapses; the project's Matrix/IRC chats peak in 2022 then fall spectacularly without a rebound, in total volume as in peer-help messages.*
- ***the decoupling** — pixls.us peer help peaks in 2023 then crumbles: the user community survives, it no longer grows, while the project's own places empty out; the deliberations, near zero before 2025, explode afterward (red dotted line), along with the module announcements, subtracted from the count and broken down below.*

Assumed limits: the volume of messages does not measure their quality — I know of no defensible metric to separate the one from the other — and the test does not bear on the cause of the decline, but on the assertion of an improvement, which no record of the project measures.

A misreading must be prevented here, for the forum's raw volumes tell the opposite story to anyone who does not

decompose them: on the surface, the Darktable category of pixls has never written as much as in 2025-2026. Decomposed, the prosperity changes nature. Peer help proper — answering someone's problem — peaked in 2023, carried by the wave of Lightroom defectors asking their way, and has been crumbling gently since. That peak is the last wave of ordinary users — photographers who were arriving and looking for their way. No wave has followed. The decline of peer help is thus not a collapse of the stock, it is a drying-up of the flow: the old-timers stay and ask fewer questions, the newcomers no longer come — and everything that grows in their place is the activity of the amateur engineers. The right word is demographic: the user base is *gentrifying* — my own surveys were already measuring it, the share of programmers among the users going from 46 to 55% between 2020 and 2022 (chapter "What insularity produces"). What grows is twofold. The module announcements and previews, first, gone from 3 to 6% of messages before 2025 to a third in 2026: the user-engineers have discovered artificial intelligence and are building their toys — neural masks, network denoising, contrast modules in preview — to the point of wearying the forum itself: in March 2026, one day apart, two threads open there — "Worried about LLM-written modules," 265 messages launched by a project contributor faced with "2 new module proposals" in one week generated by artificial intelligence, then "On module proliferation"[45]. The design deliberations, next — the terminology of AgX (a new module under construction), the interface overhauls, "Should Darktable remain AI-free?" — near nonexistent before 2025, then exploding: nearly 1,900 messages in 2025, a pace of more than 3,000 in 2026

— 10 times the level of the conflict year. Design by committee has left GitHub and ends up on the forum, as the threads of 2022 announced it would.

These deliberation threads must be read twice. As method, these deliberations are the chapter "The committee will invent nothing" in action: design choices submitted to conversation, without criteria or mandate. But as material, they are user needs — recorded without a channel, for lack of the specification that would collect them, exactly as GitHub's issues used to record needs before dying under the bot. For peer help and bug reporting measure the same thing — the care given to other people's problems, problems of understanding the tool, and problems of stability or usability of the application. That the two wither together while the toys and the mandate-less debates prosper gives the complete trajectory: the project is drifting into a fab lab of user-engineers, ever further from the software "by photographers, for photographers" that it claims to be — the photographers' needs, for their part, keep being written down; simply, no one is tasked with reading them anymore[46].

The rewriting has a police, and it operates on two terrains. The first is standing: gone, I no longer had the right to be discussed — "it's not worth to discuss about people that are not present here," "if you want to talk about Ansel with its developer, you cannot do so on pixls"; in 2025, a contributor proposed applying "a version of Godwin's law" to whoever reopened the subject[47]. My criticisms published after the departure — the video, then the article — were thus disqualified twice: on tone, and on standing, since I was no longer one of theirs, hence no longer entitled to

speak — I was also told that I could not “work in a group of people” (a criticism that appeared after my departure, never formulated before), and that one had to “stay constructive”: esprit de corps invoked as a secret to keep, in a project where everything was already public. The second terrain is content, and there the refusal was erected into policy, in writing: a moderator of the community forum declared that “the technical merits of his code, modules, vision and whatever else pale in comparison to his toxic behavior,” and the project’s principal maintainer formulated the rule: as long as the person concerned did not learn to speak to others, he “won’t be given a single though to what he has to say, period”[48]. This is the epistemic violence described above, carried to its terminus and set down in writing: assigning the speaker a credibility deficit such that the content never has to be evaluated again. Note what the policy declares exactly: not “he is wrong,” but “we will not read.” The circle does not refute; it revokes the right to be read — the tone policing described above, this time set down in writing and erected into official policy. And when I published, exhibits in hand, the technical diagnosis of the degradation — code volumes, complexity scores, extracts —, the measurable response had a remarkable structure: the implementation facts were sometimes discussed; the design questions were settled by authority; and the maintainability measures received no answer, ever — across the forum’s entire record, the words “cyclomatic,” “cognitive complexity” and “SonarCloud” do not appear a single time[49]. No one, in four years, answered a complexity figure with a complexity figure, or a worry about the project’s medium- and long-term maintainability with a tangible action meant to protect

it. The culture that calls itself direct will thus have avoided, for years on end, the most direct questions ever put to it — while remaining perfectly direct about persons. Direct about the man, evasive about the code: that is not called candor, that is called targeting.

My share remains — and it is not a fourth phase: it is a loop that ran beneath the other 3, from the first report to the rupture. My register hardened, and the breaking point is dated in the IRC/Matrix archives: May 7, 2022. After years of a perfectly civil register on the project's repository, everything tipped over the very day when, rebasing my branches, I discovered the collection-filters overhaul, which I then reported in a bug report (so improbable was the design, and so incongruous the idea that it might be deliberate). The log of the project's Matrix room documents that day hour by hour — the discovery at 12:57 p.m., the bug report at 1:26 p.m. — and it documents the sequel just as well: the loss of a patience that had already been strained since late 2021[33]. I am not one of those who endure in silence. But what hardened must be named correctly. The anger came against assumed fallacies — false, expeditious, uttered from positions of power, unassailable and nevertheless unaccountable — which I refused to tolerate any longer.

The histories of my communications on GitHub are public and show nothing but courteous exchanges with my collaborators before 2022. The IRC/Matrix logs are spicier, but that is the style of the place, and even there: no attack against anyone before late 2021. Worse, between 2019 and 2021, I organized a fundraiser in the community, as Christmas approached, to ship the maintainer

— who refused donations in money — a basket of artisanal victuals, for him and his family over the holidays. It is on the pixls.us forum that a few outbursts can be found, facing users who criticized my work without trying to understand the problems it solved, or persisted in ignoring my advice while complaining about results contrary to their expectations. Permanent exposure, in a one-against-many transaction, where anyone can afford an opinion without having studied the record, forces you to hold your patch firmly to deter the next ones from risking it. Tranquility comes at that price.

However, when years of politeness have protected nothing, improved nothing, but have on the contrary been nothing more than a mute complicity with the wrecking of the tool for the developers' fun, ceasing to muffle that anger is a refusal: an act of resistance in the sense posed above — the visible violence of the one who has nothing left but the visible means, because the situation is unacceptable by every applicable metric. The certainty that the situation was definitively locked, beyond any possible repair, was acquired exhibit by exhibit over time. Between 2020 and 2022, I watched a new habit emerge: that of features that worked reasonably well being replaced, one by one, by over-engineered horrors without any problem having been stated — the crime of the chapter "Stating the problem", serialized. A 10-year veteran said it publicly in the same terms as I did: the best Darktable he ever used was 1.4, and every release since has done nothing but decay[50] — a decline in responsiveness whose cause I would find later, while cleaning up the code in Ansel: the accumulation of ordinary programming

errors in the interface code, directly attributable to an absence of work hygiene. And when I reported these regressions, I was answered with fallacies documented as such: to defend the collection filters, the months of invested work were opposed to me — the sunk-cost fallacy of the chapter "Decide, then hold", invoked as an argument, in writing, in the project's tracker[51]. A courteous fallacy, uttered from a veto position, remains an armed fallacy — the civility of the tone only hides its dangerousness the better. When you have acquired, evidence in hand, the certainty that politeness no longer protects the project, you change objective: I was no longer trying to convince, I was trying to *deter* — to make degrading it further costly. In the Wild West described above, that was the weapon I had left. For the state of the system must be measured: the numbers, I had them — the majority vote, a core developer, the documentation maintainer, the forum moderator himself: the first phase counted the voices. And it is worse this way: in a Wild West, numbers settle nothing — access rights and attrition settle. Their free time, I did not have; and rationality had been lost.

It must also be said what my criticisms of the time were not, for it is an exhibit in the record: exhausted, overheating — the chapter "The invisible worker" has described that regime —, I confined myself, in what I published, to technical facts: code volumes, cyclomatic and cognitive complexity scores, reproducible commands[49]. Part of my diagnosis remained intuition, and I knew it; the psychological and social analysis, the one in this book, waited for distance — and for a safe place from

which to conduct it. The substance has not changed; it is simply, by now, instrumented.

This hardening was an act of resistance, and also a strategy of deterrence; judged as a strategy, its balance sheet, drawn up with the same methods as the rest, comes in two stages. The first is final: the deterrence deterred nothing — the horrors were merged anyway, for in a Wild West it is the access rights that settle matters, not the rants. I too had commit rights on the repository, and I kept them until 2026: the material capacity to undo everything, overhaul by overhaul, back to the previous state, remained in my hands for 4 years. I never used it. Undoing those overhauls would have imposed my decision by the same Wild West I denounce — access rights in place of reasons — and you do not save from itself a project whose mode of government that is. No productive, constructive way out existed in that system. The only measure equal to the problem would have been to remove the team — in a structure where the users hold a voting power, that is an option; here, nobody holds that power, and the chapter "The committee will invent nothing" said why. And leaving earlier was no more of a way out: my work was entangled in theirs, with no simple means of extracting it — defection costs all the more the more you have given, and that is why it comes so late.

The second stage of the balance sheet came later, and it changes the nature of the first. To call my strategy a "failure" would suppose that another could have succeeded. Yet nothing I reported has been fixed since — the end of this chapter will do the count — and no strategy won: politeness lost, the majority vote

lost, procedure lost — the first phase counted them. The game was locked. When all strategies fail, the problem is not the strategy.

The clearest proof came in 2026, and it does not belong to me. During the controversy over the entry of artificial intelligence into Darktable, the two men who had held the line against me — the forum moderator, the documentation maintainer — left the project in their turn. The first wrote it in black and white, to the maintainer: "Trying to speak up an make my opinion heard resulted in the maintainer telling me I was threatening the project… I've left because of your treatment and actions directly"; he details his grievances — a vote of "core devs" convened while excluding the other maintainers, his contributions treated as worthless, an accusation never followed by an apology, then silence[52]. The second withdrew without accusing anyone — he "ultimately felt it was better for the project" that he "stand aside" — but the records date his stop: his documentation merges break off the very week the decision overrode the opinion of the documentation maintainers, an exclusion the project's maintainer himself described: "The doc maintainers were against AI fine. Now I wanted to know what the devs thought about this"[52]. And on the day of their departure, the official narrative was the same as for mine: "Mica & Elstoc are no more interested in maintaining the Darktable documentation" — *no more interested*; the first replied that this was "adding insult to injury," the rewriting had not even waited[52]. Not a word about the second.

The irony does not need to be savored, for it has the value of proof. These two men reproached me for what I am, what I say and how I say it; they closed my threads, removed my links,

made my “toxicity” the official policy. Then they redid, years later and by their own path, exactly my diagnosis — a power without sharing, decisions without consultation, contributions worthless in the eyes of a single man — and they drew from it the same consequence as I did: departure. The substance is thus now common ground; only the reproach of form remains. It is the final demonstration of tone policing: when you share a man’s verdict and maintain his condemnation, it is because you only ever condemned him for his way of saying it. One would expect them in the resistance they now vindicate; they keep the gate instead — one still moderates the forum, where he himself closed the threads of the controversy; the other went quiet because it was “better for the project.” Perhaps it is Hirschman’s loyalty, the circle’s last mechanism; perhaps it is what the admission would cost — admitting that the banished man was right would oblige a retrial of the banishment. I do not arbitrate between the two: the exhibits suffice.

As for the hardened voice, it did one thing: say out loud worries I was not alone in carrying — the first phase counted them, and the chat room’s log dates them. But the same log measures why all that agreement produced nothing: the man who decided was not in the room. The merger who signs 98% of the merges — the one man who merges — wrote 36 messages there in 4 and a half years: the approval was being spent, literally, out of reach of the only person it would have been necessary to convince, and those who approved there were not going to say it again on the repository, where the deciding was done. The back-room agreement was not a safety valve by accident: it was the only

register where it cost nothing, precisely because it produced nothing there. The phenomenon has its names: the *spiral of silence* — whoever believes himself in the minority goes quiet, and his silence swells the apparent majority that will silence the next one — and *preference falsification*, the maintained gap between what one thinks in private and what one supports in public[53].

The consequence is cruel for the public dissident: the unanimity opposed to him is in part manufactured by fear, but it *looks* real, and every contester therefore seems isolated — hence eccentric, hence "toxic" — when he is sometimes the involuntary spokesman of an opinion in hiding. A forum moderator asserted that "people who like [the interface] do not voice it so loudly"; the record says better than the reverse: those who did not like it were saying so — the first phase counted them — but a critic who can be dealt with one at a time is more easily dismissed than a majority lets itself be counted, and each dismissal made the next voice more silent. And the certainty? It was accurate — provided the word is defined. What I called idiocy was never low-level incompetence: it was the inability to choose the right priorities — keeping the ergonomics aligned with desktop conventions, simplifying the code instead of replacing what works with over-engineering — and the absence of a structural understanding of their own application: a problem-solving permanently under the streetlight of the chapter "The root and the symptom", where the light is, never where the break is. The counter-examples in the record do not refute this diagnosis: they sharpen it — the honest engagements found there all bear on implementation facts, the

only level where the milieu engages; the maintainability measures and the design questions, as we have seen, never obtained an answer. One single time in the whole record, an architecture argument was advanced to justify an interface choice — and it was not even addressed to me: 3 years after my departure, in an umpteenth interface discussion on the forum, the principal merger explained to other users that the software being "plug-in based," it was difficult to control the on-screen order. The argument has the merit of being verifiable; measured on the code, it is false — the order of the modules is precisely the most centrally hard-coded thing in the whole edifice, and my own fork, measured by the same criteria, remains the same monolith, barely attenuated[54]. A strictly local intelligence: the defect is not neuronal, it is disciplinary — knowing how to code is not knowing how to design, and this entire book fits in that gap. My report was founded; left for years without an answer, it hardened into anger. The anger supplied the pretext for not reading me at all anymore. And no longer being read left me nothing but anger. Each turn tightened the next, all the way to the rupture. A system that leaves, against its own degradation, no lever but invective — then uses the invective in order not to hear about the degradation — is a system without a learning mechanism. And that is the design defect.

Four years later, the record is closed on one point: the diagnosis was right. Not because I assert it — because the records measure it, and this book has produced them chapter after chapter. The fixes decline and the issue backlog swells back up ("Ship, don't dump"); engagement melts on both sides, developers and users

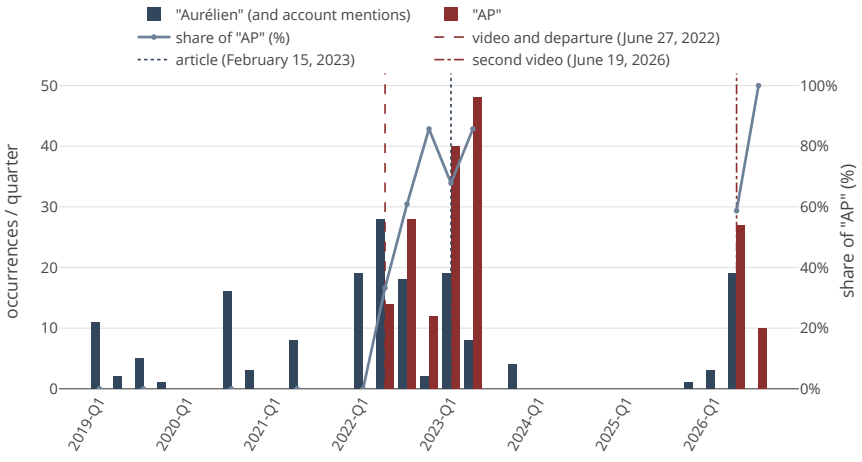
alike ("The invisible worker"); the refusal rate of pull requests is at the lowest in the project's history, while the right to merge has concentrated in a single man — 98% of merges since 2019 — who materially cannot review what he merges ("The cargo cult", "Non-heroic ambition"); and the project's places empty out while the forum drifts into a fab lab — the empty house, measured above. Even the founding promise is refuted by its own figures: the pleasure of programming was supposed to guarantee the hands, and the chapter "Working for the work" showed recruitment run dry. The project is not in good health, by the yardstick of its very own criteria.

It must finally be said who "they" is, for this word has served throughout this chapter and it would be dishonest to leave it floating. "They" is not a crowd: it is a role, held by few people — the sole merger, who decides and — the measure of it was given above — is not in the room where the talking happens; the forum's moderation core, which closes and labels; and a handful of regulars. Around them, the majority looks on, and the backstage shows that it does not always approve of what it lets happen: while the maintainer publicly defended the house, the documentation maintainer wrote in private that the current interface is "just bad" and that he could no longer stomach the endless arguments with him, which failed to prevent the degradation[55]. A passerby on the forum stated the structure better than sociology: "No one person can create a 'toxic atmosphere.' It requires the participation and complicity of several"[56]. And the immune system still runs, against data by now established: 33 threads have tried my case in my absence;

Godwin's law is proposed against whoever reopens the file; and in 2026 still, the official answer to the criticisms is the concern for image — people might come away thinking "that we are a bunch of stupid people making a crap software named Darktable"[57]. The diagnosis is confirmed; it is still not heard. A closed circle does not open when it is wrong: that is precisely how you recognize one.

The final signature of the closed circle, however, is neither the expulsion nor the rewriting: it is what it does with the expelled man's contribution. The record documents the absorption without the name: my fixes taken back into the original project, down to the commit; my estimate of the foundations worksite, redone independently 4 years later by a regular, arriving at the same orders of magnitude, without reference; my global diagnosis, endorsed at two thirds by a veteran of the project[58]. The circle can use the work of the one it banished; it cannot pronounce his name. This is not an image, it is a measurement. I counted the forms by which the forum designates me, over 1,931 messages covering 14 threads from 2019 to 2026 — those bearing on my work or on my departure[59]. For three years, before the rupture, I am called by my first name: 65 occurrences, zero initials. The first occurrence of "AP" dates from June 28, 2022 — the day after the video — and within 48 hours, 7 distinct posters adopt it; over the year that follows the departure, the initials make up nearly two thirds of the designations, and the titles of the threads themselves follow: "Interview with AP" (summer 2022), "What is it about Ansel and AP?" (2023), "AP's new video on Ansel" (2026). The mechanism is dated to the day, and it says

something more than distance: as long as I was there, people spoke to me; from the departure on, they could only speak *about* me — and to speak about the absent, the group did not keep the first name, it chose the initials. At the bottom of the slope comes the third form: "a previous developer," "that person is no longer here," "the person who designed" — the reference without a person. First name, initials, nobody: the group's language followed exactly the trajectory of the rewriting. This is not ingratitude, it is consistency: crediting the expelled would reopen the question the rewriting closed. A closed circle can be recognized by this: there, you may agree with the banished, provided you do not give the source.



How the forum names me, 2019-2026: occurrences per quarter of the first name ("Aurélien", including mentions of the account — retroactively rewritten as @anon41087856 upon the deletion of my account, at my request) against the initials ("AP"), over 1,829 messages from 14 threads devoted to my work or my departure — mine excluded; the solid line is the share of the initials among all designations (right-hand axis, plotted from 5 mentions per quarter on). 3 years at 0%; the first initials appear the day after the first video (red dashes), the share rises to 61% the following quarter and never durably comes back down — the February 2023 article (blue dotted line) revives the volume, not the form, and the second video (red dash-dots, June 19, 2026) returns to the same regime, up to 100% in the current quarter. The empty quarters are those in which no thread of the corpus is active; 2026-Q3 is partial (July).

The sociology of science has a name for this regime, and it deserves to be invoked exactly. Merton called *obliteration by*

incorporation the normal destiny of ideas that succeed: their source fades as their content gets incorporated into common knowledge — people stop citing the one whose results everyone repeats[60]. But Merton was describing a secular process, impersonal and without hostility. The project's record gives an accelerated and dated version of it, in which the erasure of the name and the reuse of the work advance in step:

Year	Share of the initials in designations (forum)	Documented reuses, the same year
2022	45% (zero over the 3 preceding years)	my videos recommended in answer to color questions (chat room, 10 occurrences)
2023	74%	"we might have to port them from Ansel" (January); videos still recommended
2024	corpus nearly silent	"maybe send him this AP video" (January)
2026	63%, then the entirety of the last mentions	commit-level backports of fixes (April); GTK 4 worksite estimate reproduced without reference; videos still cited

Mertonian obliteration takes generations; this one took 4 years, and it began the day after a video. It is not the natural course of knowledge: it is the natural course of knowledge, under pressure.

A closed circle does not reopen from the inside by good will alone: every mechanism in this chapter — the sorting by aggressiveness, the revocation of the right to be read, the rewriting, the smothering of agreement — is precisely built to neutralize individual good will. Structure is needed — but before building the structure, one must relearn what it will have to shelter: conflict itself, which is not this chapter's disease but its raw material, badly processed. Arguing is not the problem; arguing without rules is. That is the object of the next chapter.

---

1. Ferreira, I. *et al.* (2021). "The 'Shut the f**k up' Phenomenon: Characterizing Incivility in Open Source Code Review Discussions". *Proceedings of the ACM on Human-Computer Interaction*, 5(CSCW2), art. 353 https://doi.org/10.1145/3479497 — 1,545 emails from the Linux Kernel Mailing List. ↩

2. Tannen, D. (1998). *The Argument Culture*. New York: Random House https://openlibrary.org/works/OL15841541W — a general cultural frame (discourse as ritual combat), applied here to free software by analogy, not as the finding of a study about it. ↩

3. Letourneur, D. (2022). *On ne naît pas mec. Petit traité féministe sur les masculinités*. Paris: La Découverte, coll. «

Zones » https://www.editionsladecouverte.fr/on_ne_nait_pas_mec-9782355221521. The definition is taken up here for its yield, independently of the terrain on which the author deploys it: violence as the removal of options. The restriction I add to it — the removal must be arbitrary, that is, without demonstrated necessity — is mine, and it is necessary: without it, every rule would become a violence. ↩

4. Andersson, L. M. and Pearson, C. M. (1999). "Tit for Tat? The Spiraling Effect of Incivility in the Workplace". *Academy of Management Review*, 24(3), 452-471 https://doi.org/10.2307/259136, definition p. 457: "low-intensity deviant behavior with ambiguous intent to harm the target, in violation of workplace norms for mutual respect" — the ambiguity of the intent is the defining trait, and what makes the thing so difficult to sanction. ↩

5. Williams, K. D. (2007). "Ostracism". *Annual Review of Psychology*, 58, 425-452 https://doi.org/10.1146/annurev.psych.58.110405.085641 — the 4 threatened fundamental needs; Eisenberger, N. I., Lieberman, M. D. and Williams, K. D. (2003). "Does Rejection Hurt? An fMRI Study of Social Exclusion". *Science*, 302(5643), 290-292 https://doi.org/10.1126/science.1089134 — simulated social exclusion (virtual ball game) activates the anterior cingulate cortex, involved in physical pain, in proportion to the reported distress. ↩

6. Rochefort, C. (1971). « Définition de l'opprimé », presentation text for the French edition of Valerie Solanas,

SCUM Manifesto; republished as a volume, Paris: éditions les vilains, 2021, https://www.editionslesvilains.fr/livres/christianerochefort-definitiondelopprimee-texteintegral. « L'oppresseur n'entend pas ce que dit son opprimé comme un langage mais comme un bruit » ; and: « Pour l'oppresseur il n'y a pas d'oppression, forcément, mais un fait de nature. » (our translation in the body). ↩ ↩

7. Fricker, M. (2007). *Epistemic Injustice: Power and the Ethics of Knowing*. Oxford: Oxford University Press https://doi.org/10.1093/acprof:oso/9780198237907.001.0001. Fricker reserves the concept for prejudices aimed at social groups; its application to a credibility deficit constructed individually is a borrowing of structure, flagged as such. ↩

8. Imran, M. M. & Sarker, J. (2025). "'Silent Is Not Actually Silent': An Investigation of Toxicity on Bug Report Discussion". *FSE 2025, Ideas, Visions and Reflections* track https://arxiv.org/abs/2503.10072 — qualitative analysis of 203 GitHub bug-report threads, 81 of them toxic. Toxicity there arises mainly from misaligned perceptions of the bug's severity and priority, from unresolved frustrations with the tool, and from lapses in professional communication; these exchanges "reduce the likelihood of actionable outcomes, such as linking issues with pull requests". A preliminary study, on a sample deliberately enriched in toxic threads. ↩

9. Sarker, J., Turzo, A. K. & Bosu, A. (2025). "The Landscape of Toxicity: An Empirical Investigation of Toxicity on GitHub". *Proceedings of the ACM on Software Engineering* (FSE 2025)

https://doi.org/10.1145/3715744 — 2,828 projects sampled, automatic classification by a toxicity detector specialized for software engineering. A project's popularity is positively associated with the prevalence of toxicity, its issue-resolution rate negatively; the authors of toxic messages reoffend significantly more, and themselves become targets more often — which joins the regime described here, where everyone is by turns shooter and target. Measured associations, not established causality. ↩

10. Johnson, A. (2017). "The Multiple Harms of Sea Lions", in *Perspectives on Harmful Speech Online*. Cambridge, MA: Berkman Klein Center, Harvard University https://cyber.harvard.edu/publications/2017/08/harmfulspeech — the tactic "fuses persistent questioning—often about basic information, information easily found elsewhere, or unrelated or tangential points—with a loudly-insisted-upon commitment to reasonable debate" and "works both to exhaust a target's patience, attention, and communicative effort, and to portray the target as unreasonable". The term comes from a comic strip: Malki, D. (2014). *Wondermark*, No. 1062, "The Terrible Sea Lion" https://wondermark.com/1k62/ . ↩

11. Deneault, A. (2015). *La Médiocratie*. Montréal: Lux https://openlibrary.org/works/OL27260738W — in the original French: « Il n'y a eu aucune prise de la Bastille, rien de comparable à l'incendie du Reichstag, et l'Aurore n'a encore tiré aucun coup de feu. Pourtant, l'assaut a bel et bien été lancé et couronné de succès : les médiocres ont pris

le pouvoir » ; « Rangez ces ouvrages compliqués, les livres comptables feront l'affaire » (wordings verified against the publisher's edition; our translations in the body). ↩

12. Bailey, A. (2018). "On Anger, Silence, and Epistemic Injustice". *Royal Institute of Philosophy Supplement*, 84, 93-115 https://doi.org/10.1017/s1358246118000565 ; Cloud, D. L. (2021). "The Rhetoric of Civility as Soft Repression", in *Civility, Free Speech, and Academic Freedom in Higher Education*. London: Routledge. https://doi.org/10.4324/9780429282041-9 ↩

13. On the construction of madness as a "female malady" and the psychiatric pathologization of women's deviance: Showalter, E. (1985). *The Female Malady: Women, Madness, and English Culture, 1830-1980*. New York: Pantheon. https://openlibrary.org/books/OL3023442M/The_female_malady On the dismissal of women's anger by diagnosis — discarding the content by qualifying the form: Chemaly, S. (2018). *Rage Becomes Her: The Power of Women's Anger*. New York: Atria. https://www.simonandschuster.com/books/Rage-Becomes-Her/Soraya-Chemaly/9781501189562 ↩

14. Lorde, A. (1981). "The Uses of Anger: Women Responding to Racism", address to the National Women's Studies Association, June 1981; reprinted in *Sister Outsider*. Trumansburg, NY: The Crossing Press, 1984, pp. 124-133 https://archive.org/details/sisteroutsideresoolord : "Tell me how you feel but don't say it too harshly or I cannot hear you. But is it my manner that keeps her from hearing, or the

threat of a message that her life may change?"; "Anger is loaded with information and energy". On tone policing as silencing: Bailey, A. (2018). "On Anger, Silence, and Epistemic Injustice". *Royal Institute of Philosophy Supplement*, 84, 93-115 https://doi.org/10.1017/s1358246118000565. ↩

15. The fundamental attribution error: Ross, L. (1977). "The Intuitive Psychologist and His Shortcomings". *Advances in Experimental Social Psychology*, 10, 173-220 DOI; the actor-observer asymmetry: Jones, E. E. and Nisbett, R. E. (1971). *The Actor and the Observer*. Morristown, NJ: General Learning Press https://openlibrary.org/works/OL10486732W. Malle, B. F. (2006). "The Actor-Observer Asymmetry in Attribution: A (Surprising) Meta-Analysis". *Psychological Bulletin*, 132(6), 895-919 https://doi.org/10.1037/0033-2909.132.6.895 — across 173 studies, the general asymmetry is close to zero, but it subsists for negative events (d ≈ 0.24, brought down toward 0.11 after correction for publication bias) and reverses for positive ones: a self-serving pattern — which is precisely the regime of the blame distributions discussed here. ↩

16. Becker, H. S. (1963). *Outsiders: Studies in the Sociology of Deviance*. New York: Free Press of Glencoe, p. 9 https://openlibrary.org/works/OL44024314W — quoted from the original edition. ↩

17. Baron, R. A. (1988). "Negative Effects of Destructive Criticism: Impact on Conflict, Self-Efficacy, and Task Performance". *Journal of Applied Psychology*, 73(2), pp. 199-

207 https://doi.org/10.1037/0021-9010.73.2.199 — destructive criticism is defined there by 3 traits: non-specific, inconsiderate in tone, and imputing the poor performance to *internal* rather than situational causes. Recipients report more anger and tension, and state that they will handle subsequent disagreements by resistance or avoidance rather than by collaboration or compromise. This is the construct that software-engineering research took up to measure destructive code reviews, cited further on. ↩ ↩

18. Torvalds, L. (September 16, 2018). "Linux 4.19-rc4 released, an apology, and a maintainership note", Linux kernel mailing list https://lkml.org/lkml/2018/9/16/167 : "I need to change some of my behavior, and I want to apologize to the people that my personal behavior hurt and possibly drove away from kernel development entirely." In the wake, the project replaces its *Code of Conflict* with a code of conduct derived from the *Contributor Covenant* — "Code, conflict, and conduct", *LWN.net*, September 2018 https://lwn.net/Articles/765108/ ; text in force: Contributor Covenant Code of Conduct https://docs.kernel.org/process/code-of-conduct.html . ↩

19. Tourani, P., Adams, B. & Serebrenik, A. (2017). "Code of Conduct in Open Source Projects". *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)* https://doi.org/10.1109/SANER.2017.7884606 — the first empirical study of the codes of conduct of free software: they serve as much to prevent as to sanction, but their

application is uneven, and nothing establishes that on their own they ease the arrival of newcomers. ↩

20. Hanika, J. & Ellinghaus, T. (June 2, 2019). “10 years of Darktable”, *Libre Graphics Meeting 2019*, Saarbrücken, Sunday June 2, 10:50 a.m. — official recording published by LGM, https://www.youtube.com/watch?v=04cnoNMdZBw (28 min 57 s, posted online July 17, 2019): the slide titled “do-ocracy” appears at 17 min 31 s. The program summary announced a history of the project, then a discussion of the “different approaches of team organisation” and the lessons drawn from 10 years of a changing group https://libregraphicsmeeting.org/2019/sunday-02-06/index.html. Text of the slide: “if there is a rule, it’s don’t stop anyone from coding when they’re having fun”. ↩ ↩ ↩

21. Do-ocracy is catalogued as a governance model in its own right: those who do the work make the decisions, authority being proportioned to the time and energy invested in a given domain. On these informal governances, their functioning and their instability, see Kostakis, V., Niaros, V. & Giotitsas, C. (2015). “Production and Governance in Hackerspaces: A Manifestation of Informalism?”. *International Journal of Cultural Studies*, 18(5), pp. 555-573 https://doi.org/10.1177/1367877913519310. ↩

22. Schneider, N. (April 29, 2020). “How’s That Open Source Governance Working for You?”, Media Economies Design Lab, University of Colorado Boulder https://www.colorado.edu/lab/medlab/2020/04/29/hows-open-source-governance-working-you — “basic tools for

real accountability—the elections, the juries, the term limits —are nowhere to be found". The author does not treat do-ocracy under that name, but the informal governances of free software in general, whose defect he explicitly ties to the tyranny of structurelessness. ↩

23. Same talk ([20]), next slide, at 18 min 38 s, titled "responsibility": "whether we like it or not: people depend on our software to get their work done". The content given to it is the promise of indefinite compatibility of past edits — the commitment that the sidecar file of a photograph, reopened years later, reproduce the same result. A promise kept, and costly: it constrains the software's architecture, as the chapter "Code is a liability" shows. ↩

24. Same talk ([20]), "lessons learned" slide, at 20 min 28 s: "gained what from laissez-faire? 1. quick development, 2. features I never thought of, 3. translations I never knew existed, 4. technical debt :/". The term *laissez-faire* is used by the speakers themselves to designate their organizational model. ↩

25. Gunawardena, S. D. *et al.* (2022). "Destructive Criticism in Software Code Review Impacts Inclusion". *Proceedings of the ACM on Human-Computer Interaction*, 6(CSCW2), art. 292 https://doi.org/10.1145/3555183 — destructive criticism demotivates all respondents, but disproportionately increases the exit intention of women and non-binary developers. ↩

26. Terrell, J. *et al.* (2017). “Gender differences and bias in open source: pull request acceptance of women versus men”. *PeerJ Computer Science*, 3:e111 https://doi.org/10.7717/peerj-cs.111 — careful wording: the acceptance rate of women’s contributions is higher overall, but drops below men’s when the gender becomes identifiable. ↩

27. Miller, C. *et al.* (2022). “‘Did You Miss My Comment or What?’ Understanding Toxicity in Open Source Discussions”. *Proceedings of ICSE 2022* (Distinguished Paper) https://doi.org/10.1145/3510003.3510111 — across 100 toxic GitHub threads, the dominant motif is the demanding user against the maintainer. ↩

28. Thread “Does the world need another incomplete photo editor?”, discuss.pixls.us, March 6, 2016 https://discuss.pixls.us/t/does-the-world-need-another-incomplete-photo-editor/975 : the request — “improve the stability and speed of existing software, without introducing yet more ‘options’… Make the existing software more accessible”; the answers — a core developer: “Freeloaders are a burden and not an asset”; a moderator: “most of which are based on ignorance”. ↩

29. Inventory of thread closures via the forum’s public API, July 2026. The Darktable category of the main forum counts only about twenty closed threads in 10 years: apart from technical duplicates, the threads of interface or direction criticism are massively over-represented there. Examples, with the closing verbatim: “Would a configuration wizard help new

users of Darktable?" (October 2025, closed in two days — "Well, that's enough I think"); "Color balance RGB chroma slider value range [POLL]" (December 2022, two days — "people will either take the survey or they won't"); "Devs - How Hard Would It Be To…?" (January 2023, 7 days); "Another dt UI discussion" (June 2026, 246 messages, 4 days — "this has run its course"). Of the 16 closures after heating up recorded from 2016 to 2026, 10 of the 13 documented down to the last message are closed by the same moderator, with a recurring vocabulary ("run its course", "that's enough"), generally two to 9 days after the opening. These threads really were tense at the moment of their closure — but the tension arises precisely where the criticism of the interface meets the defense of the project, and the closure settles the tension, not the criticism. ↩

30. The two-month suspension: thread "Is development really important?", August 2021 ("Enough is Enough", with the reminder that the forum is "not a free speech zone"). The formula "nobody is forcing you" is attested from January 2018 to 2026, five times out of seven from the same moderator's pen. ↩

31. PR #10694 (February 2022, ≈600 lines) against PR #11066 https://github.com/Darktable-org/Darktable/pull/11066 (February-April 2022, +7,631 lines): the earlier work crushed, and the question of the duplicate, put in writing, left without any answer in the record. ↩

32. Chronology on the project's tracker, April-May 2022. PR #11066 https://github.com/Darktable-

org/Darktable/pull/11066 merged on April 17 (8 participants), the "recently used collections" module deleted the same day. The opposition: a dozen tickets in three weeks, including #11757 https://github.com/Darktable-org/Darktable/issues/11757 (opened May 7 by the documentation maintainer; 89 comments, 14 participants; still open in 2026) — the core developer: "Current way is definitely not readable or intuitive"; the community lead: "my desire would be to revert the GUI pieces to what they are in the current 3.8.1". The revert: PR #11752 https://github.com/Darktable-org/Darktable/pull/11752, vote 6 for / 4 against, refused by the maintainer: "Let me even put this straight… I'm quite opposed to this PR"; then, in #11757: "Well I cannot force you to be happy :)" and "I'm always in favor of moving forward". The grounds, in #11747 https://github.com/Darktable-org/Darktable/issues/11747 (closed the day of its opening, labeled "controversial"): "I started this huge work months ago… And now that it has land into master, at less than 1 week from freeze… no, we can't go back, I'm sorry" (the author of the overhaul); "To be fair most people don't like changing or moving forward… That's remind me all the haters about the new GNOME 3.0" (the maintainer); "I know kids scream from what they want until they have it, not adults" (#11757); "but stay constructive". The concession: PR #11801 https://github.com/Darktable-org/Darktable/pull/11801, merged in 77 minutes on May 14; the 4.0 release notes https://github.com/Darktable-org/Darktable/releases/tag/release-4.0.0 announce the

overhaul without mentioning the fallback preference. The documentation maintainer's ticket was closed on May 25 on a tooltip, against his advice; he reopened it on December 6. ↩

33. Element export of the public Matrix room `#Darktable` (bridge of the IRC channel), July 23, 2026, 118,996 messages from February 2022 to July 2026 — in my possession, verifiable by joining the room. On May 7, 2022: "WHAT ?" (12:57 p.m.), "oh fucking shit, progress got in the way of productivity once again" (12:58 p.m.), link to the bug report (1:26 p.m.). The same log establishes my own register of August 2022 — volume at its highest in May (2,880 messages), invective in August — and the exchange of August 8 analyzed in the chapter "Listening without obeying": the exhibit charges both camps, and it is on that basis that I file it in the record. ↩ ↩ ↩ ↩ ↩ ↩ ↩ ↩ ↩ ↩

34. Ticket #12319 https://github.com/Darktable-org/Darktable/issues/12319, "Define and implement a process for large changes to the project", opened on August 13, 2022 by the community lead (7 approvals), citing by name "the collection filter changes" among the cases to be framed; closed on August 18: "Alright. We're already doing the best we can then." ↩

35. #11757 https://github.com/Darktable-org/Darktable/issues/11757, December 6, 2022, a core developer: "Imho the biggest mistake was to not keep the old one as default" — and the principal defender of the overhaul: "you're probably right… keeping old widget by default

would have been better". Same thread, July 30, 2026: "Then the originator and me got shut up :-)". ↩

36. Hirschman, A. O. (1970). *Exit, Voice, and Loyalty*. Cambridge, MA: Harvard University Press https://openlibrary.org/works/OL2745855W — when voice is rendered inoperative, exit remains; the fork is the pure form of software exit. ↩

37. Log of the Matrix chat room ([33]), verbatims: the documentation maintainer, May 7, 2022 — "it felt like a dev scratching an itch" — and May 25, 2022 — "wow Pascal thought the tooltip was *the* fix to the readability problems and closed my issue"; the week of the departure: "Thanks for the work and the rant!" (June 27), "It looks like a bit of a worry" (June 29, about the new interface of the main branch). In public, on the forum: "AP chose to use those words as is his right" (https://discuss.pixls.us/t/31388/31, June 28). The chat room's supporters, though, never relayed their agreement on the repository or the public forum. ↩

38. The video: published June 27, 2022, discussed the same day on https://discuss.pixls.us/t/31388 (opened by the user vbs: "the language was a bit too strong … I think there are a lot of good points in the video"; archived: https://web.archive.org/web/20260723092719/https://discuss.pixls.us/t/r-Darktable-options-and-path-forward-and-user-point-of-view/31388); its link was removed from the thread by the moderators for "very strong language". The article: "Darktable: crashing into the wall in slow-motion", https://ansel.photos/en/news/Darktable-dans-le-mur-au-

ralenti/ , February 15, 2023 (the site's header date; French original archived: https://web.archive.org/web/20260720234454/https://ansel.photos/fr/news/Darktable-dans-le-mur-au-ralenti/) — about 10,000 words, with sources and examples in support. The absence of an answer on substance is documented quantitatively further on in this chapter (the 3-level analysis). ↩

39. Same speaker — paperdigits, forum administrator —, discuss.pixls.us: 2022-07-04, "Nobody has been rejected" (https://discuss.pixls.us/t/31388/153 ; archived: https://web.archive.org/web/20260723092318/https://discuss.pixls.us/t/r-Darktable-options-and-path-forward-and-user-point-of-view/31388/153) and 2022-06-29, "he asked to be removed and I obliged him" (https://discuss.pixls.us/t/31388/54 ; archived: https://web.archive.org/web/20260723094014/https://discuss.pixls.us/t/r-Darktable-options-and-path-forward-and-user-point-of-view/31388/54); 2026-06-20, "it is our fault (particularly mine) for tolerating such poor behavior for so long" (https://discuss.pixls.us/t/58735/10 ; archived: https://web.archive.org/web/20260721230927/https://discuss.pixls.us/t/aps-new-video-on-ansel/58735/10); "thrived since he left": https://discuss.pixls.us/t/58735/49 (archived: https://web.archive.org/web/20260723092430/https://discuss.pixls.us/t/aps-new-video-on-ansel/58735/49). The Matrix log ([33]) dates the gap between the two registers of the same speaker to within 3 days: on July 1, 2022, he writes to me in the chat room "that's why you'll end up by yourself" and

calls my video “fucking pathetic and sad”; on July 4, on the public forum: “Nobody has been rejected”. The register of the chat rooms is cruder than that of the forums for everyone, me included; what the pair establishes is not a character trait, it is the gap between the stage and the backstage at the very moment the official narrative was forming. ↩

40. Festinger, L. (1957). *A Theory of Cognitive Dissonance*. Evanston: Row, Peterson https://doi.org/10.1515/9781503620766; Halbwachs, M. (1950). *La mémoire collective*. Paris: PUF (critical ed. Albin Michel, 1997) https://openlibrary.org/works/OL1232451W — the group’s memory rebuilds itself in the service of its present. ↩

41. Verbatim, discuss.pixls.us, 2022-07-03: “Mostly I’m hearing the equivalent to good riddance! I find it astonishing that the person who has brought Darktable to a completely new level in terms of color processing and scene referred workflow feels like he has to leave the project” (https://discuss.pixls.us/t/31388/150, user anon16817310; archived: https://web.archive.org/web/20260722032615/https://discuss.pixls.us/t/r-Darktable-options-and-path-forward-and-user-point-of-view/31388/150). A message from a third-party user, contemporaneous with the departure, contradicting in real time the official “nobody has been rejected” of the same week. ↩

42. Verbatim, discuss.pixls.us, 2023-03-26: “This has all the look and feel of a mob attack, one where the participants are intoxicated breathing their own exhaust” (https://discuss.pixls.us/t/33910/198, user meaningfullyhappy; archived: https://web.archive.org/web/20260722032643/https://discuss.pixls.us/t/ansel-was-r-Darktable/33910/198). A message from a third party describing the thread in progress. ↩

43. Leymann, H. (1990). “Mobbing and Psychological Terror at Workplaces”. *Violence and Victims*, 5(2), 119-126 https://doi.org/10.1891/0886-6708.5.2.119; and (1996) “The Content and Development of Mobbing at Work”, *European Journal of Work and Organizational Psychology*, 5(2), 165-184 https://doi.org/10.1080/13594329608414853. In French: Hirigoyen, M.-F. (1998). *Le harcèlement moral : la violence perverse au quotidien*. Paris : Syros. https://openlibrary.org/works/OL572454W ↩

44. Contributors: chapter “Ship, don’t dump”, deduplicated git history. Chat room: Matrix log ([33]) — messages from the other members, monthly average over March-August 2022 (my presence): 6,786; from October 2022 to March 2023: 3,174 (−53%); late 2023: about 1,200 (−82%). Peer help: addressed answers (quotations and nominal callouts, method of the chapter “The invisible worker”) — 4,905 in 2022, 1,585 in 2023, about 1,500 a year since. The trough is never followed by a rebound, which “thriving” would require. Bound of the test: it bears on the project’s own places — its chat room, its repository, its peer help —, not on the whole

ecosystem. The third-party French-language forum (Darktable.fr, 5,101 threads counted via the API), in decline since its 2018 peak for causes prior to the conflict — the migration of the public toward the English-language forum —, shows on the contrary a rebound in 2026. Corrected for the noise of the introduction threads the forum imposes on new members: about 245 content threads at an annual pace against 150 in 2025 (+63%), and the introductions themselves run at nearly 4 times the 2025 pace — an influx of new members. I report it because the test must also report what goes back up. ↩

45. "Worried about LLM-written modules" https://discuss.pixls.us/t/worried-about-llm-written-modules/56430, March 4, 2026, 265 messages — the opening, by a project contributor: "Within a week, we've seen 2 new module proposals, both created by people who made it clear… that they had an idea, and used an LLM to quickly come up with an implementation"; "On module proliferation" https://discuss.pixls.us/t/on-module-proliferation/56435, March 5, 2026, 102 messages. The toys named in the body have their own threads, the same weeks: "Contrast Management RGB: A new scene-referred approach (POC)", 359 messages; "[New module] Color harmonizer", 228 messages. ↩

46. Genre decomposition of the threads: title keywords, then manual review of the 4 largest quarters and the 25 largest threads, 48 threads reclassified one by one — including a second audit pass on the peak quarters of 2023 and 2024,

which moved only 3 percent of their messages: those peaks really are peer help — lists, rules and identifiers in the corpus. Among the reclassified: “Editing moments” (1,322 messages, photography showcase), “current OSX Build” (1,004, build coordination), “Should Darktable remain AI-free?” (724, governance), “New Sigmoid Scene to Display mapping” (808, module development). Peer help cleaned up, my messages excluded: 4,155 messages in 2019, 7,937, 8,455, 10,114, then the peak in 2023 (10,491), 9,119, 8,985, about 9,700 at the 2026 pace — growth until 2023, carried notably by the questions of migration from Lightroom, then a slow erosion. The design deliberations: between 21 and 340 messages a year from 2020 to 2024, then 1,893 in 2025 and a pace of more than 3,200 in 2026. The announcements and previews: 3 to 6% of messages before 2025, a third in 2026; “On module proliferation”, 2026, documents the forum’s weariness toward its own makers. ↩

47. April 2023, community forum: “it’s not worth to discuss about people that are not present here”; “if you want to talk about Ansel with its developer, you cannot do so on pixls”. November 2025: “I would be in favour of the moderators applying a version of Godwin’s law whenever someone tries to bring this whole thing up again”. “Work in a group”: July 2022, in answer to a request for a point-by-point response — “this guy his really good at making nice modules but cannot work in a group of people” (sic). ↩ ↩

48. Verbatim, discuss.pixls.us, 2023-04-02: “The technical merits of his code, modules, vision and whatever else pale in

comparison to his toxic behavior" (https://discuss.pixls.us/t/36370/2 ; archived: https://web.archive.org/web/20260723092528/https://discuss.pixls.us/t/we-have-a-problem-what-is-it-about-ansel-and-ap-edit-concern-fully-resolved-via-respectful-dialogue/36370/2); and Darktable-org/Darktable, PR #12233, 2022-08-21: "until Aurélien learn to speak to others I won't be given a single though to what he has to say, period" (https://github.com/Darktable-org/Darktable/pull/12233 ; archived: https://web.archive.org/web/20260722015957/https://github.com/Darktable-org/Darktable/pull/12233). The quantitative analysis of the responses — implementation facts sometimes discussed, maintainability measures never (zero occurrences of "cyclomatic" or "sonarcloud" in the forum's history), design settled by authority — bears on 25 sourced assertions confronted with more than 2,000 messages. ↩

49. Record published in 2022-2023: code volumes, counted SQL queries, SonarCloud scores — for example, cyclomatic complexity of the library of interface controls: 735, technical debt estimated at 1 day 7 hours; after rewriting in Ansel: 494 (https://ansel.photos/en/news/unDarktable-ing-gui-controls/). Across the entire record of the community forum, the terms "cyclomatic", "cognitive complexity" and "sonarcloud" do not appear a single time: the figures were never discussed there, nor taken up, nor contested. ↩ ↩

50. Verbatim, discuss.pixls.us, 2023-03-20 (https://discuss.pixls.us/t/33910/85 , user LebedevRI): "The

best Darktable i ever used was Darktable-1.4. Every release, every day since, it has been actively and gradually decaying." (archived: https://web.archive.org/web/20260723092117/https://discuss.pixls.us/t/ansel-was-r-Darktable/33910/85). The same message as the endorsement of the 60-80% quoted further down. ↩

51. Darktable-org/Darktable, issue #11747, 2022-05-07 — answer from the author of the overhaul (AlicVB): "I started this huge work months ago. everything has been transparent… And now that it has land into master, at less than 1 week from freeze… no, we can't go back, I'm sorry." The invested work and the calendar stand in for an argument there; the validity of the criticism is not discussed. (https://github.com/Darktable-org/Darktable/issues/11747; archived: https://web.archive.org/web/20260723092220/https://github.com/Darktable-org/Darktable/issues/11747) ↩

52. The controversy: PR #20322 ("AI inference subsystem", February-March 2026) and the thread "Should Darktable remain AI-free?" https://discuss.pixls.us/t/52778 (724 messages, locked). The community lead: "Trying to speak up an make my opinion heard resulted in the maintainer telling me I was threatening the project" and "I've left because of your treatment and actions directly" (messages #516 and #530 https://discuss.pixls.us/t/52778/516); the three grievances, including the vote of "core devs" excluding the other maintainers: #556 https://discuss.pixls.us/t/52778/556;

the silence: #600 https://discuss.pixls.us/t/52778/600; on GitHub, after the vote: “When a project let’s you know that your contributions are worthless, listen to them” (PR #20322, February 18, 2026). The maintainer, facing him: “He told me clearly that if AI was introduced in Darktable he will quit the project. That’s just plain threatening” (#529 https://discuss.pixls.us/t/52778/529); and “You’ve expressed your opinion (and elstoc also)… The doc maintainers were against AI fine. Now I wanted to know what the devs thought about this” (#557 https://discuss.pixls.us/t/52778/557). The withdrawal of the documentation maintainer: “ultimately felt it was better for the project for me to stand aside” (thread “New dtdocs maintainer(s) needed”, message #3 https://discuss.pixls.us/t/57489/3); his merges on the documentation repository stop on February 9, 2026 — the litigious PR opens on the 11th — and resume, by other hands, only on May 10. The announcement, by the maintainer: “Mica & Elstoc are no more interested in maintaining the Darktable documentation” (#1 https://discuss.pixls.us/t/57489/1); the community lead’s reply: “it was your combination of action and inaction… from your position of power as maintainer of the project that lead to me leaving the project… But adding insult to injury just seems to be how you do things” (#2 https://discuss.pixls.us/t/57489/2). ↩ ↩ ↩

53. Noelle-Neumann, E. (1974). “The Spiral of Silence: A Theory of Public Opinion”. *Journal of Communication*, 24(2), 43-51 https://doi.org/10.1111/j.1460-2466.1974.tb00367.x; Kuran, T.

(1995). *Private Truths, Public Lies: The Social Consequences of Preference Falsification*. Cambridge, MA: Harvard University Press https://openlibrary.org/works/OL3527602W . The IRC testimony falls under the regime of the chapter "Listening without obeying" (channel not archived); the maintainer's quotation ("people who like it do not voice it so loudly") is, for its part, public: https://discuss.pixls.us/t/58799/228 (archived: https://web.archive.org/web/20260723093046/https://discuss.pixls.us/t/another-dt-ui-discussion/58799/228). ↩

54. The argument: "Darktable is plug-in based… every module is self contained and does not know about other plug-ins. So it is harder to have specific control order on screen" (discuss.pixls.us, thread "Another dt UI discussion", message 22, June 2026 https://discuss.pixls.us/t/58799/22). Code audit conducted on July 22, 2026 on the local repository, pre-fork version (May 24, 2022) compared with Ansel's current version, same metrics. Darktable: 718 references to module names outside the modules directory, spread across 31 files; `src/common/iop_order.c` (2,211 lines) contains on its own 3 complete, hard-coded orderings of the full set of modules — the order does not "result" from plug-ins that ignore one another, it is a central table the core consults permanently. The monolith subsists in Ansel, by the same criteria, with a smaller amplitude (hardcoding of module groups brought from 245 entries down to 0, contact with the global structure from 98% of files down to 93%). ↩

55. The project's Matrix room (private export, cf. [33]), the documentation maintainer: March 20, 2023, "this made the ui look cluttered"; March 23, 2023, "I can't stomach the endless arguments with pascal, which often end up with worse UX as he tries to satisfy everyone"; January 13, 2024, "current interface is just bad". In public, the maintainer, June 26, 2026 (https://discuss.pixls.us/t/58799/228 and 230 https://discuss.pixls.us/t/58799/230): "I do like a lot the way the UI is… The Darktable devs like the way the UI is." ↩

56. discuss.pixls.us, thread 33910, messages 198 and 202 https://discuss.pixls.us/t/33910/198, March 26, 2023: "This has all the look and feel of a mob attack"; "No one person can create a 'toxic atmosphere.' It requires the participation and complicity of several… if that 'atmosphere is still here' in the absence of the person being blamed for 'creating it', it confirms… the complicity and participation of those others". ↩

57. Inventory via the forum's API: 33 threads posterior to the departure trying my case outside my presence. Godwin's law: cf. [47]. The image, July 10, 2026: "People not knowing well Darktable and Ansel could think when reading AP post that we are a bunch of stupid people making a crap software named Darktable. This is just plain wrong." ↩

58. Reuses with attribution: Darktable-org/Darktable, PR #20811 https://github.com/Darktable-org/Darktable/pull/20811 and #20798 https://github.com/Darktable-org/Darktable/pull/20798 (April 2026), backports of Ansel

fixes down to the commit — an outcome that the insiders were anticipating as early as January 2023, in the project's Matrix room: "we might have to port them from Ansel" (January 9), "I think pretty much any commits from AP will be cherry-picked from ansel" (March 17 — log cited in [33]). Endorsement: "i fully and wholeheartedly agree with 60%..80% of what is said in that post" (https://discuss.pixls.us/t/33910/85, 2023-03-20, archived: https://web.archive.org/web/20260723092117/https://discuss.pixls.us/t/ansel-was-r-Darktable/33910/85; a 10-year contributor — today the maintainer of a third-party library and himself in disagreement with the current governance: corroboration from an aligned critic, not an internal verdict). GTK 4 estimate redone in 2026 (≈40,000 lines, 18-24 months) concurring with my 2023 figures, without reference. ↩ ↩

59. Count by script over the forum's public API, July 23, 2026: 14 threads (2019-2026) bearing on my work or my departure, 1,931 messages, nested quotations excluded so as not to count the same text twice, and my own messages (102 of 1,931) excluded from the count. Before June 27, 2022: 65 nominal designations ("Aurélien", "Aurelien", or a mention of the account), 0 "AP". From June 27, 2022 to June 30, 2023: 75 nominal, 142 "AP" — 65%. In 2026, the share holds above half; the current (partial) half-year no longer contains anything but initials. First occurrence: https://discuss.pixls.us/t/31388/3, June 28, 2022. A known bias, unfavorable to the measurement: the anonymization of my account — at my own request, under the GDPR — has retroactively rewritten the @aurelienpierre mentions as

@anon41087856, counted here as nominal whatever their date. Typing economy does not explain the switch: the abbreviation was available for 3 years and did not serve a single time before the conflict. The nameless forms (“a previous developer”, “that person is no longer here”: https://discuss.pixls.us/t/54423/138, 2026-01-06; “the person who designed”: https://discuss.pixls.us/t/58799/243, 2026-06-26) are too rare for a statistic and stand as qualitative markers. Independent replication: the log of the project’s Matrix room ([33]) shows the same switch — as long as I am there, the first name dominates by far (623 mentions against 34 in the second quarter of 2022); the quarter in which I leave the room, the switch (8 against 48 in the 4th quarter), then the initials reign for good. Two places, two corpora, the same fold at the same moment. ↩

60. Merton, R. K. (1968). *Social Theory and Social Structure*, enlarged ed. New York: Free Press, pp. 25-38 https://openlibrary.org/works/OL269872W; verbatim definition in Merton, R. K. (1988). “The Matthew Effect in Science, II”. *Isis*, 79(4), p. 622 https://doi.org/10.1086/354848: “the obliteration of the sources of ideas, methods, or findings by their being anonymously incorporated in current canonical knowledge”. Merton describes a normal process of science, slow and without hostility; applying it to a case where the erasure of the name is contemporaneous with a conflict is my extension, and I flag it as such. The reuses in the table are sourced in notes [58] and [33]. ↩

28. The well-tempered conflict

After the preceding chapter, the temptation is known in advance: less conflict. Pacified communities, muffled disagreements, benevolence in all things. That is not just terribly naive, it is a misreading, and it needs correcting. The closed circle was not sick from an excess of conflict; it was sick from conflicts *without rules* — settled by means, never on the merits — and, just as much, from conflicts *prevented*: the spiral of silence that deprived disagreement of a public voice was a pacifying machine, and it produced the worst. The absence of dispute is not peace: it is often the sign that someone has already won without having had to debate. This chapter writes the rules that were missing — for any project that would rather learn from its conflicts than burn in them.

People who care about something fight for it: conflict arises when people have irreconcilable interests, judge the matter important enough to commit to it, and have the resources to do so. It is a healthy process: through it you assert your will, you weigh on your own future. Refusing it because it would be "violent" is submitting in advance to the most powerful party: the one that wins without fighting, by sheer inertia, waiting for the others to wear out. Yet the chapter "Three-tiered violence" defined violence by what it produces — suffering — not by the noise it makes. A status quo that grinds people down in silence is violent. The noisy fight contesting it is not necessarily so. A community

without any visible dispute should worry, not reassure: either no one there cares about anything anymore, or those who care have already been silenced — and the preceding chapter showed which of the two it was.

Organizational research confirms this rehabilitation, and sharpens it with a capital distinction: there is conflict, and there is conflict. *Task* conflict — about the substance: this problem, this solution, these criteria — improves decisions; organized controversy, where opposing positions instruct each other, produces better judgments than spontaneous unanimity[1]. *Relationship* conflict — aimed at the other person, their worth, their intentions — degrades everything it touches. And the most useful result of this literature is their relation: task conflict bears fruit only where relationship conflict is contained, that is, where an idea can be attacked without its author feeling attacked — the psychological safety of the chapter "The committee will invent nothing", once again[2]. The whole art, then, is to hold both gestures at once: heat the debate on the objects, cool it on the people. The chapter "The closed circle" showed the symmetrical double failure: a milieu that tolerated attacks on persons, and refused debate on the merits.

Now, this book spent its first half building, without saying so, a machine for converting relationship conflicts into task conflicts. Take up the apparatus again in that light. The problem stated in writing (chapter "Stating the problem"): as long as the problem is implicit, opposing it means opposing someone; written down, it becomes an object that can be attacked without touching anyone. The validation criteria set before the solution: they move the

dispute from "your idea against mine" to "which one satisfies the criteria" — and closure (chapter "Decide, then hold") gives the conflict a possible end, what the chapter "The word and the thing" called the stopping rule. Measurement (chapters "Measuring without lying to yourself" and "Counting the absent"): an auditable number is an impartial third party in miniature, the only witness both camps can cross-examine. Climbing back to needs (chapters "The root and the symptom" and "Stating the problem"): most conflicts over solutions are conflicts over positions, and dissolve when you climb back to the needs — the dinner and the movie of the chapter "Stating the problem". Testimony against prescription (chapter "Listening without obeying"): it disarms the conflict of authority by giving everyone a domain where they are sovereign and a domain where they are not. None of this eliminates conflict: all of it *civilizes* it — in the literal sense, it moves it from the duel to the trial.

The same apparatus, read in negative, finally explains why the conflicts of free software are so uniformly *experienced* as conflicts between persons — and why they become just that. In a project without a specification, a design disagreement literally has no object to latch onto. No written goal: opposing a proposal is opposing its author. No arbitrated priority: each camp invokes its own. No definition of what "optimal" means: optimal in speed versus optimal in readability, optimal for my workflow versus optimal for yours, with no text saying which one the project promised. Deprived of an object, the disagreement falls back on the only object available: the persons. It is not that characters are

worse there than elsewhere; it is that the deficit of technical method converts mechanically into relational damage — the exact mirror of the chapter "There is no technical problem": there is no technical problem, but an absent technical method manufactures human ones. And the loop feeds itself: once relations are damaged, proposing to finally write down the goals and the criteria looks like a maneuver by the opposing camp — the method requires precisely the trust its absence has destroyed. A conflict without a resolution criterion does not get resolved; it *ends*, and it can only end by domination: seniority brought to bear, merge rights, the veto — the chapter "The closed circle" has documented that regime. As for the users, their place in this theater says it all: they eat popcorn. The discussion threads bear witness — dozens of spectators follow the controversy like a soap opera, take sides for characters, never for criteria, and for good reason: there exists no criterion they could invoke. In a project with a specification, they would be the natural referees — their needs, on record, would decide between the camps; it is even one of the functions of a specification to give the conflict a judge who is none of the parties. Without it, the referee's seat is empty, and that empty seat is what de facto power comes to occupy.

The empty seat does not only call in de facto power; it calls in a substitute norm. Lacking a criterion to sort true from false, a community can police only what everyone knows how to see: tone. The correctness of a claim requires an expertise and an arbitration it does not have; the rudeness of a reply, anyone can observe. So it sanctions the only thing it knows how to sanction, and lets the other pass: confident error, provided it is polite. This

division is not peculiar to software. The disciplines whose errors kill — medicine, civil engineering — gave themselves codes, standards and a duty of care precisely because error kills; those whose errors kill no one, like image processing, have nothing of the kind, and the absence of sanction reads there as a license for negligence[3]. But the error that does not kill still costs: it misleads, it misinforms, it makes everyone lose time that never comes back — the user will have to unlearn what he thought he knew, and the designer will have to defuse the falsehood before he can even explain his tool. A disagreement of fact, without a metric of truth, therefore cannot close on the facts: it falls back on tone, where the only punishable thing is register. Tone becomes the judge the community does not have — and a perverse judge, who protects polite error and condemns blunt correctness.

Before drawing up the trial, one case must be settled that the Wild West produces in series, and that traps something true. Designing in open source means designing under permanent public scrutiny: every detail will be criticized, always, in waves, at the wrong moment, often by people who have not read the thread and never will. That rolling fire is exhausting, and the designer has a legitimate need to shelter from it — this book has measured the fatigue enough not to dispute it. But the legitimate shield degenerates into immunity, and I will give a dated case, in which I am a party. In May 2022, I reported in a bug report the regressions of the collection-filters overhaul; the author of the overhaul closed his reply thus: “I’m sorry to be so rude, but I daily face ’that was better before' remarks and injunction in my real work… and that’s just nonsense… done by people that (on

purpose or not) ignore the reality and how the real world has changed"[4]. The reply is fallacious twice over, and it is worth taking the time to establish it, because the generic mechanism is what matters. What reality had changed? Not photography — neither as a trade nor as a practice; nothing in the real world had required this overhaul. The only thing that had changed was the tool, and it had changed because its author had changed it: invoking "the real world" to cover one's own decision is passing off a design choice as a law of nature — and note that the exhaustion invoked comes from elsewhere, from his job, imported into the exchange to disqualify an interlocutor who had nothing to do with it. Second fallacy: the criticism did not come from a passing stranger, but from a developer of the project's core, a longtime teammate on the same repository. And had it come from a stranger, its substance would have weighed no less — for it can be measured. The old filtering function: 548 lines, 22 functions, average cyclomatic complexity of 1.9, no function above the ceiling the metric recommends. The system that replaced it: 5,207 lines spread over 16 files, 147 functions, average complexity of 4.7, including two functions beyond that ceiling[5]. Tastes can be debated; a factor of 10 in code volume and a factor of two and a half in complexity, to replace something that existed, worked, and let itself be understood, cannot. The shield against harassment served here to push back an argued bug report: this is immunity through exhaustion — the fatigue, real, converted into a permanent exemption from substance.

The other face of the same shield is the heroic narrative. I have read, in the French-speaking developers' Matrix chat room, from

the pen of the maintainer who has merged the bulk of the code for years, the admission that it is sometimes hard to make every decision alone, then take the reproaches afterwards — I quote from memory[6]. The admission is human, and it calls for two known ways out: delegate responsibilities, or consult those the decision affects before making it. The merge record, in the chapter "Thinking against oneself", shows that neither was taken — the concentration has grown year after year. So the third way remains, the way of the narrative: the figure of the necessarily lonely chief, who carries the burden and endures the ingratitude. That figure has a precise function: it keeps the power and dodges accountability — solitude is inverted into legitimacy instead of being treated as the symptom of an organizational defect. And it feeds itself: because he is alone, he decides alone; because he decides alone, the reproaches rain down afterwards; because they rain down, the chief's solitude is confirmed, and the circle closes into hagiography. You do not get out of it through psychology; you get out of it through structure.

These two figures — immunity through exhaustion, solitude as vocation — are the same defect seen from two positions. The designer's need for protection is legitimate; the user's need for expression is too; and the Wild West makes them destroy each other. Without an instituted channel, substantive criticism has only one way in: pressure — insistence, the wave, the pillory — that is, what borders on harassment; the system encourages precisely what it complains about, because it is the only route that sometimes gets an answer. And it is not even a route: the defense is symmetrical to the attack, chapter "The closed circle". A

system under permanent tension, with no organized discharge, needs a fuse — failing which it ends up as a church: behaviors where nothing is rational anymore, nothing is falsifiable, and where people no longer debate, they commune[7] — the chapter "The cargo cult" described that regime, and the chapter "The word and the thing" said why an unfalsifiable design is no longer a design. The fuse bears a less exalted name than freedom: institutions.

There remains what the trial requires and the Wild West does not have: institutions, however tiny. Elinor Ostrom, studying the commons that survive across centuries, made it one of her principles: durable communities all have *rapid and low-cost* conflict-resolution mechanisms — no grand tribunals, accessible arenas where disputes get settled before they fester[8]. Transposed to a project, even a small one, it fits in four tiers that can be written on one page. Self-settlement first: the structured discussion of the chapter "The committee will invent nothing" — one topic, one place, off-topic moved out, a deadline. Mediation next: a third party accepted by both camps, who does not decide but forces each side to restate the other's position until it is recognized — half of all conflicts die there, for lack of combatants, when each discovers what the other really meant. Arbitration after that: someone, designated in advance and for a limited time, decides on the merits, in writing, criteria in support — and the decision applies, revisable through the channels of the chapter "Decide, then hold": new evidence, or nothing. The vote last, as a final resort, for the questions of course that belong to the collective — never for design, chapter "The committee will invent

nothing". Four tiers, one page; what matters is not their sophistication, it is that they exist *before* the conflict. A protocol written cold protects everyone; a protocol improvised in the heat of the moment is just one more weapon in the Wild West. A formula heard along the way, whose author I never traced, sums it all up: democracy is agreeing on protocols for disagreement. Political science says the same thing in treaty language. Democracy does not resolve conflicts, it processes them. Purely procedural rules are enough to structure and contain them. And that is what distinguishes civil peace from peace by force[9].

To which add three hygiene rules that my experience makes it necessary to write. Treat conflicts *early*: the chapter "The closed circle" tells of years of grievances accumulated in private then discharged all at once; a conflict handled within the week is a conversation, the same conflict handled within the year is a war — and conflict debt runs at the same interest rate as technical debt. Protect the dissenter, not just the majority: the countermeasures of the chapter "The committee will invent nothing" — anonymous votes, devil's advocate, instituted dissent — exist precisely so that disagreement does not have to choose between silence and heroism; a community where it takes courage to disagree is already sick. And apply the rules symmetrically: the record of the preceding chapter shows what becomes of a moderation that weighs the critic's words and not his adversaries' — a police force perceived as a faction ceases to be a police force, and every asymmetrical sanction manufactures resentment that will resurface elsewhere, uglier.

The objection is economic, and it is made in good faith: all this is bureaucracy, a small project cannot afford procedures, we will see when we are grown up. Two answers. The first is bookkeeping: the cost of unmanaged conflict is documented in this book — years of guerrilla warfare, a burnout, a fork, a community cut in two; next to that, one page of protocol and an occasional mediator are free. The second is empirical: the commons that last *all* have their conflict arenas, however small — it is not a luxury of large organizations, it is a survival condition observed over centuries; and the recent history of free software says the same thing in the negative, every major project having ended up giving itself, in crisis, the institutions it refused in calm. The rule of the Wild West is that you die young there. So the question is not whether a project will have conflicts — it will, and it must, for a project without dispute is a project nobody cares about anymore. The question is whether its conflicts will feed it or devour it, and that depends almost not at all on the people: it depends on the rules written beforehand. But before writing those rules, one last obstacle must be lifted, and it is the deepest: a milieu does not give itself conflict arenas as long as it believes it does not need them — as long as its self-image serves as its balance sheet. That image must therefore be examined, and with it the gap between what free software says about itself and what its own archives record. That is the object of the next chapter.

---

1. "Constructive controversy": Tjosvold, D. (1985). "Implications of Controversy Research for Management". *Journal of Management*, 11(3), 21-37

https://doi.org/10.1177/014920638501100303 ; in the tradition of Deutsch, M. (1973). *The Resolution of Conflict: Constructive and Destructive Processes*. New Haven: Yale University Press https://doi.org/10.12987/9780300159356 — the father of the distinction between cooperative and competitive conflict. ↩

2. Jehn, K. A. (1995). "A Multimethod Examination of the Benefits and Detriments of Intragroup Conflict". *Administrative Science Quarterly*, 40(2), 256-282 https://doi.org/10.2307/2393638 — task conflict versus relationship conflict. The meta-analysis by De Dreu, C. K. W. & Weingart, L. R. (2003), *Journal of Applied Psychology*, 88(4), 741-749 https://doi.org/10.1037/0021-9010.88.4.741 , shows that even task conflict harms performance when it is not contained — which strengthens, rather than weakens, the demand for rules and psychological safety made here (Edmondson, chapter "The committee will invent nothing"). ↩

3. I develop this division in "Image processing does not kill people… and it's a shame" (2021): the technical fields capable of harm — medicine, civil engineering — give themselves an apparatus of diligence (codes of ethics, standards, licenses to practice) that fields without vital risk dispense with, which "provides an excuse for negligence, lack of rigor and overall laziness"; and bad tools "do cognitive damage that will need work later to be reversed", at the cost of everyone's time. https://eng.aurelienpierre.com/2021/02/image-processing-does-not-kill-people-and-its-a-shame/ ↩

4. Darktable-org/Darktable, issue #11747 https://github.com/Darktable-org/Darktable/issues/11747, May 7, 2022 — postscript of the reply by the author of the overhaul (AlicVB), quoted verbatim in the body (archived: https://web.archive.org/web/20260723092220/https://github.com/Darktable-org/Darktable/issues/11747). The same message invokes the work invested and the imminent freeze to refuse any rollback — an angle analyzed in the chapter "The closed circle". The initial bug report is mine. ↩

5. Cyclomatic complexity counts the linearly independent execution paths of a function — every branch, condition or loop adds one; in practice it is the minimal number of test cases needed to cover the logic, and a measure of what a reviewer must hold in mind at once (McCabe, T. J. (1976). "A Complexity Measure". *IEEE Transactions on Software Engineering*, SE-2(4), 308-320 https://doi.org/10.1109/tse.1976.233837). McCabe proposed capping it at 10 per function — "a reasonable, but not magical, upper limit" — and the NIST guide admits up to 15 only for seasoned teams, with organized reviews and a complete test plan (Watson, A. H. and McCabe, T. J. (1996). *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. NIST Special Publication 500-235 https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-235.pdf, § 2.5, p. 15). The threshold of 15 is also that of the measurement tool used here (lizard): the "ceiling" of the body text is therefore the permissive bound, not the

strict one. Measurements of July 23, 2026 on the public repository: at tag release-3.8.0, src/libs/tools/filter.c has 548 lines; at release-4.0.0, its replacement totals 5,207 lines (filtering.c: 2,072; 16 files under filters/: 3,135), the old file being reduced to 127 lines of delegation. Averages per function: 1.9 over 22 functions, none above 15, versus 4.7 over 147 functions, two above. The collect.c module, which the overhaul did not replace, went in the same interval from 3,317 to 3,588 lines. ↩

6. Darktable-fr Matrix chat room, discussion channel of the French-speaking developers, not publicly archived: the testimony falls under the regime of the chapter "Listening without obeying" and I quote it from memory. What does not depend on memory: the public merge record of the chapter "Thinking against oneself" — from some fifteen pull-request mergers in the early 2010s to 2, a single one signing 95 to 100% of the merges since 2019 — and the review arithmetic of the chapter "The cargo cult". ↩

7. The ritual can be documented because it is formulaic — collected through the main forum's public API, July 2026. The same formulas recur over 8 years, whatever the grievance. "Nobody is forcing you": 5 occurrences from the same moderator, from 2018 to 2026. "Fork it": a RawTherapee developer, 2021 — "we don't care much about public opinion… they can fork it and change it, or avoid it" (thread "Exposure tone curves profiles", message 9 https://discuss.pixls.us/t/exposure-tone-curves-profiles-per-brand-or-camera/23367/9); a Darktable regular, the same

year, condenses the prove-it / fork-it / pay-someone triptych into a single reply (thread “Why do CAT and filmic render clipped areas so weirdly”, message 81 https://discuss.pixls.us/t/solved-why-do-cat-and-filmic-render-clipped-areas-so-weirdly/26426/81). Two threads show the complete regime. “v4.4 appears grayed-out” (2024) https://discuss.pixls.us/t/v4-4-appears-grayed-out/40694 : a user reports that the default theme is unreadable; a core developer replies that this gray is “a design decision for the best of our user’s experience” — irrefutable by construction — and invites him to stop posting (“or maybe you are just a troll!”); Darktable 5.0 would ship a higher-contrast theme. “DorS Control Naming Refinement Proposal” (2026) https://discuss.pixls.us/t/dors-control-naming-refinement-proposal/56400 : a contributor proposes renaming a module’s sliders, does the work, opens a PR. Regulars counter him with the authority of the absent author: “The author of the module (AP) knew what he was doing”. That authority leans on my own article about the module’s mathematics — a falsifiable text, that one, but one that says nothing about slider names. The PR would be withdrawn. A gray declared irrefutable; a criticism validated by the facts, expelled all the same; the writing of a banished man brandished as dogma on a question it does not settle: the liturgy is complete. The same forum knows how to do otherwise: a 2023 thread comparing Darktable to Lightroom received some forty replies of pure technique and not a single dismissal. The ritual is dominant, not exclusive. ↩

8. Ostrom, E. (1990). *Governing the Commons*. Cambridge: Cambridge University Press https://doi.org/10.1017/cbo9780511807763 — 6th design principle of durable commons: conflict-resolution mechanisms that are rapid, low-cost and locally accessible. ↩

9. Przeworski, A. (2011). "Divided We Stand? Democracy as a Method of Processing Conflicts". *Scandinavian Political Studies*, 34(2), 168-182 https://doi.org/10.1111/j.1467-9477.2011.00265.x — Johan Skytte Prize lecture, 2010. The thesis, in three steps: civil peace has almost always rested on the threat of force; yet conflicts can be "structured, regulated and contained"; and "purely procedural rules can be effective in processing conflicts". From the same author, the shortest definition of the regime: "democracy is a system in which parties lose elections" — *Democracy and the Market* (1991), Cambridge: Cambridge University Press https://doi.org/10.1017/cbo9781139172493. The protocols are fixed, the outcomes are not. ↩

29. Thinking against oneself

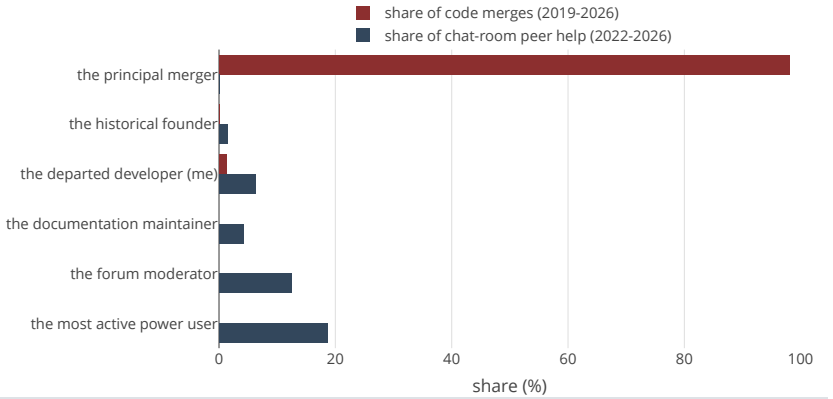
The previous chapter established that conflicts need rules; this one asks why an entire milieu has lived without them for 40 years, and the answer lies in its mirror. Free software tells a story about itself: open, welcoming, meritocratic, self-correcting, leaderless. Every one of those words can be checked, because this milieu has a property unique in the world — everything is archived. Every forum, every repository, every moderation decision leaves a public, dated trace. A movement that claims to live by transparency thus offers to scrutiny what no institution offers: the possibility of comparing, exhibit by exhibit, what it says about itself with what its own logs record. The test does not stop at manners. Organizations profess values they do not always practice: organization theory has distinguished, for half a century, the *espoused theory* — what a group declares it believes and wants — from the *theory-in-use* its acts reveal, and the gap between the two is the rule rather than the exception, often without the actor even being aware of it[1]. Free software gives that gap a unique test bench. "Darktable is created for photographers, by photographers," proclaims the project's home page — a claim about *whom* it serves, as checkable as the others, and one the measured composition of the milieu refutes (chapter "What insularity produces"): created for developers, by developers. The diagnosis dictates the remedy: a milieu that never thinks against itself cannot correct itself, and the chapter "The pact" will propose the institution it lacks.

The first article of faith, and the most famous: "given enough eyeballs, all bugs are shallow." The line is not Torvalds's — Raymond coined it and christened it "Linus's Law," and Torvalds himself immediately weakened it, noting that the one who finds a problem is almost never the one who understands it[2]. The weakened version is reasonable; the slogan version has been falsified by the cleanest experiment imaginable. The OpenSSL encryption library secured an enormous share of the web: millions of servers, all the eyeballs in the world available. A memory-read flaw was introduced there in late December 2011, shipped in March 2012, and discovered in April 2014 — more than two years of exposure in the most strategic code on the Internet[3]. And the lesson is not where it usually gets put: the infrastructure was not watched by a crowd, it was carried by *one* full-time developer, in a foundation that received about $2,000 in donations a year. The project's treasurer wrote it the very week of the crisis: "the mystery is not that a few overworked volunteers missed this bug; the mystery is why it hasn't happened more often"[3]. The dogma's eyeballs do not exist: nobody reviews encryption code for free on a Saturday. Nadia Eghbal generalized the finding by taking inventory of free software's actual production: "open source" describes how code is *distributed and consumed* — it says nothing about how it is *produced*; production, measurements in hand, rests almost everywhere on one or two people in front of a stadium of spectators[4]. The bazaar is a consumers' legend. The cathedral never stopped being built by 3 workmen — unpaid.

Second article: meritocracy. Its first known printed use, in 1956, is pejorative; the book that launched it, two years later, is a *dystopia* — a satire in which the society of merit ends up overthrown by revolt, and whose author, Michael Young, spent his final year begging people to stop brandishing as a badge the word he had invented to frighten[5]. Free software adopted it as a coat of arms without reading the book: cargo cult, chapter "The cargo cult", applied this time to a word. That leaves the content of the blazon to test. Its local version comes down to one dogma: only the code counts, the patch speaks for itself, never mind who signs it. Now that dogma is measurable, and it is false. At comparable contribution, patches from reputed developers are reviewed faster and succeed more often than those from unknowns; and evaluators consult the author's profile and social ties before judging their code[6]. What gets called merit is therefore already, in part, status. Ethnography then showed what the dogma does to the rest. Dawn Nafus put the free-software communities under the microscope; her finding: "openness" there is applied to the code and denied to the social. That partition delegitimizes the work through which a newcomer, whoever they are, acquires competence: welcoming, transmission, mentoring. Nafus's field was the exclusion of women — hence her title, "patches don't have gender," the dogma quoted verbatim — but the mechanism she describes does not choose its victims: it pushes aside anyone not already inside[7]. The chapter "What insularity produces" gave the numbers of what this produces; here we understand the mechanism that protects them: a screen-word, never tested, that converts every measurement of exclusion into a finding of merit.

Third article, and the deepest: "no leaders, no power." The chapter "The committee will invent nothing" delivered Freeman's verdict — the absence of structure does not abolish power, it makes it invisible. Now for the genealogy and the proof. The genealogy: Robert Michels, 1911, studying the parties that claimed the purest democracy, drew from them what he himself named the "iron law of oligarchy" — who says organization, says a *tendency* to oligarchy: the elected settle in, the apparatus closes on itself, and power slips away from the very people in whose name everything is done[8]. The proof, a century later, on the very terrain that claimed to escape it: Shaw and Hill analyzed the complete history of 683 large wikis — peer production in its chemically pure form — and tested the iron law through 3 hypotheses: the more a community grows, the fewer new administrators it appoints; the more administrators concentrate governance activity; the more experienced contributors get reverted by them. All 3 hold[9]. And I do not need to borrow anyone else's terrain: the Darktable repository records its own iron law. The chapter "The cargo cult" measured the tightening; here is the complete series: 24 people were merging code in 2016, 8 in 2019, two or three since 2021 — and a single developer has signed, every year since 2019, between 89 and 100% of all merges. Everything that enters the software passes through the hands of one man — no mandate, no term, no appeal, and nobody ever decided it would be so: the organization secreted it, which is exactly Michels's thesis[10]. The conclusion to draw is not cynical, it is architectural: since oligarchy grows *by itself*, the choice was never between power and no power — it is between accountable power and unaccountable power. Free software chose without

knowing it: commit rights without a mandate, moderators without appeal, vetoes without expiry. The Wild West of the chapter "The closed circle" was not an anomaly; it was the iron law, operating in the dark.



Who decides, who cares: share of the repository's code merges (2019-2026, 9,316 pull-request merges, merged_by field, red) against share of peer help in the project's chat room (2022-2026, 9,025 addressed replies, blue), by role. Two different bases, a single question — the one of the chapter "Taking care": do the hands that hold the decision hold the care? The principal merger signs 98% of the merges and 4 peer-help replies in four and a half years; peer help lives with those who decide nothing. The same merger answered on the forum until 2019, among the top 6, before withdrawing from it; and merging would be a form of care too — if it came with review, which the chapter "The cargo cult" measured.

Then comes the question of the mirror, and it can be measured. I had the professed principles and the recorded acts collected from

the public archives of the forum and of the project this book studies — keeping only the institutional gaps, never the persons, and recording the counter-examples too, because there are some[11]. Three exhibits suffice here. The forum's posted rule orders users to "criticize ideas, not people" and never to answer the tone rather than the substance; the case file of the chapter "The closed circle" showed the actual regime — the substance never discussed, the tone discussed without end. The project's website still invites, today, in the same terms as in 2022: "Just use it! And afterwards let us know how you like it. Send your bug reports, feature requests, suggestions or just your personal opinion to our mailing list." The archives show what those who did exactly that received — the accusation of trolling for a theme defect fixed two releases later, the "fork it and implement the missing stuff yourself" for the others. And the day a link was hidden by moderation, the official explanation was "it was not censored" — in the very thread whose technical log records the act of hiding, timestamped, signed, then its lifting the next day[11]. This last case is the most instructive precisely because it is small: the link was restored within a day, a moderator even publicly congratulated himself on it — the system *knows how* to backtrack. What it does not know how to do is see itself: not one of these gaps was ever noted *from the inside*. Counter-examples included, the professed values were real enough to be invoked, and they were enforced — asymmetrically: with constancy against critics and outsiders, rarely against the group, never against those charged with enforcing them.

Where does the asymmetry come from? Not from malice: from a group mechanism that psychology named and described half a century ago. Irving Janis, studying the decision fiascos of American general staffs, isolated *groupthink*: when cohesion becomes the supreme value, the group starts producing unanimity instead of producing judgment. Its symptoms read like an index to this book's case file: the illusion of unanimity — the chapter "The closed circle" showed the spiral of silence that manufactures it; self-censorship — the IRC dissenters who disapproved in private and kept quiet in public; direct pressure on whoever doubts; and the "mindguards," the word is Janis's — built on the model of the bodyguard: self-appointed members who protect the group not from blows, but from information that would shake its confidence in itself[12]. Reread the moderation of the chapter "The closed circle" with that word in mind. And then take the measure of the irony, for it is the key to the chapter: this movement was born to fight real dominations — the proprietary vendor that dictates to users what they are allowed to do, the spyware that harvests their data. It won those fights in the code. And it recreated, in its forums, the exact structure it was fighting: a power that dictates the admissible beliefs, polices conformity and extinguishes dissent — the dictatorship driven out of the code, reinstalled in the conversation. What the group asks of its members is neither a need to serve nor care to give — the chapter "What insularity produces" showed that those words are missing from the ethos; what it asks is conformity: love the workflow, praise the tool, adopt the jargon, and applaud novelty at the pace of the economy of acclamation.

What do the institutions that avoid this trap have? They are not staffed with better people; they are *equipped against themselves*. Science is the canonical example, and it must be read correctly. Merton described its ethos in 4 norms, the last of which, *organized skepticism*, says everything in its adjective: doubt there is not an individual virtue, it is an institutional mandate — "the scientific investigator does not preserve the cleavage between the sacred and the profane, between that which requires uncritical respect and that which can be objectively analyzed"[13]. Nothing, not even the founding beliefs of the institution, is exempt from examination. Popper drew the consequence that upends common sense: scientific objectivity does not result — *cannot* result — from a researcher's efforts to be objective; it results from the "friendly-hostile co-operation" of many scientists, that is, from institutions: peer review, replication, the right of refutation[14]. An open society, in Popper's sense, is not a society without beliefs: it is a society whose beliefs remain attackable from the inside. Free software opened its code and closed its culture — everything about the artifact is inspectable, nothing about the institution is: no moderation audit, no appeal body, no public accounting of conflicts, not one single inquiry the milieu ever inflicted on itself. Its only organ of correction is the fork — that is, exit. Now Hirschman demonstrated the vice of that choice: easy exit does not improve the system, it anesthetizes it, because those who care most about quality — "the most active, reliable, and creative agents of voice" — are precisely the ones who leave first, and their departure deprives internal criticism of its carriers[15]. "Fork it" is not a policy of correction: it is the outsourcing of conscience. And the case file of the chapter "The closed circle"

shows the gesture that completes the circuit: the forker, conscience outsourced, is then excommunicated — the milieu rids itself of its critique by asking him to leave, all while keeping his contributions, then congratulates itself on hearing him no more.

It remains to say what true openness would be, and that is the thesis of this chapter. The movement opened the code — which is to say, it opened power to those who can read it. The chapter "Freedom in a void" put numbers on what that condition excludes: the vast majority of users, for whom "the freedom to modify the program" is a clause written in a language they will never speak. You cannot demand that users read the code; you can open something else to them: *governance*. Seats, mandates, minutes, a right of appeal — a social structure built on top of the open code, which has remained insularity's main blind spot precisely because insularity never had to think about it: among peers who read code, technical openness suffices. This is not an unprecedented utopia; it is a tradition free software never looked at. In the early 1980s — before the GNU manifesto — the UTOPIA project brought together in Stockholm researchers and the Nordic print workers' union: typographers, 120,000 members behind them, were co-designing in a "technology laboratory" their own page-layout tools, not because they knew how to program, but because the work was theirs — participatory design, since codified into an academic discipline, demonstrated 40 years ago that non-programmers can govern the design of their tools[16]. Irony has it that these were printing workers: the very trade from which free software inherited its fonts, its cases and its vocabulary. So here is the next opening, and it is not technical: to

extend to users not the text of the program, but the government of its design — and for the milieu, to learn at last to think against itself, that is, to stop being a club and become an institution. The concrete form of that institution — who sits, who pays, who decides, who answers — occupies the two chapters that close this part: the "incompetent worker", where the house learns to recognize and to bound the know-how it lacks without driving anyone out; then the "pact" that founds it.

---

1. Argyris, C. & Schön, D. A. (1974). *Theory in Practice: Increasing Professional Effectiveness*. San Francisco: Jossey-Bass — the distinction between *espoused theory* (what an actor declares they believe and pursue) and *theory-in-use* (what their acts actually reveal), and the finding that the gap between the two is ordinary and often unnoticed by the actor himself. ↩

2. Raymond, E. S. (1999). *The Cathedral and the Bazaar*. Sebastopol, CA: O'Reilly (first presented at Linux Kongress, Würzburg, May 27, 1997) http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ . Lesson 8: "Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone"; "Or, less formally, 'Given enough eyeballs, all bugs are shallow.' I dub this: 'Linus's Law'." Raymond himself records Torvalds's objection: the one who finds the problem is not, in general, the one who understands it. ↩

3. CVE-2014-0160 (“Heartbleed”) https://nvd.nist.gov/vuln/detail/CVE-2014-0160: flaw introduced on December 31, 2011, shipped in OpenSSL 1.0.1 on March 14, 2012, disclosed on April 7, 2014. The financial testimony: Marquess, S. (2014). “Of Money, Responsibility, and Pride”. veridicalsystems.com, April 12, 2014 (archived) http://web.archive.org/web/20140415221341/http://veridicalsystems.com:80/blog/of-money-responsibility-and-pride/ — about $2,000 in donations a year; a single team member without an outside full-time job (Stephen Henson); “there should be at least a half dozen full time OpenSSL team members, not just one”; “the mystery is not that a few overworked volunteers missed this bug; the mystery is why it hasn’t happened more often”. The foundation otherwise lived on commercial contracts (under a million dollars gross a year), tied to deliverables, not to maintenance of the core. ↩ ↩

4. Eghbal, N. (2020). *Working in Public: The Making and Maintenance of Open Source Software*. San Francisco: Stripe Press https://press.stripe.com/working-in-public: “The term ‘open source’ refers only to how code is distributed and consumed. It says nothing about how code is produced”. She reports there that a GitHub study attributes, in more than 85% of the projects examined, over 95% of the code and interactions to fewer than 5% of the developers; and her “stadium” typology: many users, almost no contributors. Upstream: Eghbal, N. (2016). *Roads and Bridges: The Unseen Labor Behind Our Digital Infrastructure*. Ford Foundation

https://www.fordfoundation.org/learning/library/research-reports/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure/ . ↩

5. First recorded printed use (and the *Oxford English Dictionary*'s first citation): Fox, A. (1956). "Class and Equality". *Socialist Commentary*, May 1956 — already pejorative. The book: Young, M. (1958). *The Rise of the Meritocracy, 1870-2033: An Essay on Education and Equality*. London: Thames & Hudson https://archive.org/details/riseofmeritocracooooyoun — a dystopian satire ending in revolt against the elite of merit. The disavowal: Young, M. (2001). "Down with meritocracy". *The Guardian*, June 29, 2001 https://www.theguardian.com/politics/2001/jun/29/comment — "I have been sadly disappointed by my 1958 book"; Young laments the adoption of the word, all the way into Tony Blair's speeches, as a positive ideal. ↩

6. Bosu, A. and Carver, J. C. (2014). "Impact of Developer Reputation on Code Review Outcomes in OSS Projects: An Empirical Investigation". *Proceedings of ESEM 2014*, ACM/IEEE https://doi.org/10.1145/2652524.2652544 — across 8 major free-software projects: patches from core developers receive a first response sooner and complete their review earlier than those from peripheral contributors. On the use of social signals in evaluation: Marlow, J., Dabbish, L. and Herbsleb, J. (2013). "Impression Formation in Online Peer Production". *Proceedings of CSCW 2013*, 117-128 https://doi.org/10.1145/2441776.2441792 ; and Tsay, J.,

Dabbish, L. and Herbsleb, J. (2014). "Influence of Social and Technical Factors for Evaluating Contribution in GitHub". *Proceedings of ICSE 2014* https://doi.org/10.1145/2568225.2568315 — acceptance of a contribution rises with the social proximity of its author to the project, technical factors held constant. ↩

7. Nafus, D. (2012). "'Patches don't have gender': What is not open in open source software". *New Media & Society*, 14(4), 669-683 https://doi.org/10.1177/1461444811422887 — "openness" is analyzed there as a construction that separates the technical from the social: men "monopolize code authorship and simultaneously de-legitimize the kinds of social ties necessary to build mechanisms for women's inclusion". ↩

8. Michels, R. (1911). *Zur Soziologie des Parteiwesens in der modernen Demokratie*. Leipzig: Klinkhardt https://archive.org/details/zursoziologiedeoomichgoog (English translation: *Political Parties: A Sociological Study of the Oligarchical Tendencies of Modern Democracy*, trans. E. and C. Paul, 1915). The unmodalized formula ("who says organization, says oligarchy", 1915 English edition, p. 401) is harsher than the German original, which speaks of a *tendency*; it is the modalized version that is quoted here. ↩

9. Shaw, A. & Hill, B. M. (2014). "Laboratories of Oligarchy? How the Iron Law Extends to Peer Production". *Journal of Communication*, 64(2), 215-238 https://doi.org/10.1111/jcom.12082 — the complete internal history of 683 of the largest wikis of the Wikia platform,

2004-2010; the 3 hypotheses (rarer appointments of administrators, concentration of governance activity, growing reversions of experienced contributors) all three hold: "we find support for Michels' iron law". Limits stated by the authors: a single platform, oligarchy operationalized through administrative behavior. ↩

10. Merged pull requests of the Darktable-org/Darktable repository, exhaustive survey through the API (`merged_by` field), 2012-2026: 11, 15, 16, 13, 11, 8, 6, 5, 6, 4, 3, 3, 2, 3 then 2 distinct mergers per year; share of the top two: 34 to 65% from 2012 to 2015, 70% in 2016, 99% from 2019 on, 100% to within one unit since 2023. An earlier measurement on merge commits alone gave the same profile with higher headcounts before 2019 (up to 24): local branch merges, without pull requests, counted there as merges — the pull-request measurement is the right one for the question asked, which is the right to decide on other people's contributions. In a sample of 21 pull requests merged in 2026, all 21 were merged by the same developer. The chapter "The cargo cult" does the arithmetic of what this bottleneck makes impossible. ↩

11. Ledger built from public archives (the forum's Discourse API, action logs included; https://www.Darktable.org/development/ , wording verified identical between the Wayback snapshot of July 3, 2022 and the live 2026 page, re-archived: https://web.archive.org/web/20260613134148/https://www.Darktable.org/development/). The 3 exhibits: forum rules,

https://discuss.pixls.us/faq ("criticize ideas, not people"); the project's invitation and its actual reception, including https://discuss.pixls.us/t/40694 (accusation of trolling for a theme report, later fixed; archived: https://web.archive.org/web/20260714060151/https://discuss.pixls.us/t/v4-4-appears-grayed-out/40694); the logged hiding, https://discuss.pixls.us/t/interview-with-ap/31924 (archived: https://web.archive.org/web/20250513031948/https://discuss.pixls.us/t/interview-with-ap/31924) — `action_code: visible.disabled` (July 31, 2022), "it was not censored" (August 1, 2022), hiding lifted the same day. The counter-examples are recorded in the same ledger: a swift and applauded restoration, one moderation intervention in favor of the critic, credited code reuses; and the forum never claimed to be a free-speech platform — its operators said so bluntly, which bounds the accusation to what is documented. ↩ ↩

12. Janis, I. L. (1972). *Victims of Groupthink*. Boston: Houghton Mifflin https://openlibrary.org/books/OL21279371M/Victims_of_groupthink — the "mindguard", pp. 41-42: "Just as a bodyguard protects the President and other high officials from injurious physical assaults, a mindguard protects them from thoughts that might damage their confidence in the soundness of the policies to which they are committed or to which they are about to commit themselves"; the typology of the 8 symptoms in 3 families: Janis, I. L. (1982). *Groupthink*, 2nd

ed. Boston: Houghton Mifflin, pp. 174-175 https://openlibrary.org/books/OL3513757M/Groupthink. ↩

13. Merton, R. K. (1942). "Science and Technology in a Democratic Order". *Journal of Legal and Political Sociology*, 1, 115-126; reprinted as "The Normative Structure of Science", in *The Sociology of Science*. Chicago: University of Chicago Press, 1973, chap. 13, pp. 267-278 https://archive.org/details/sociologyofscienoooomert_j7p1 — the 4 norms (universalism, "communism" of findings, disinterestedness, organized skepticism), p. 270; the sacred and the profane, pp. 277-278. ↩

14. Popper, K. R. (1945). *The Open Society and Its Enemies*, vol. II, chap. 23 ("The Sociology of Knowledge"). London: Routledge https://www.routledge.com/The-Open-Society-and-its-Enemies-Hegel-and-Marx/Popper/p/book/9780415278423 — "science and scientific objectivity do not (and cannot) result from the attempts of an individual scientist to be 'objective', but from the friendly-hostile co-operation of many scientists" (pp. 217-218 of the revised Routledge edition). ↩

15. Hirschman, A. O. (1970). *Exit, Voice, and Loyalty*. Cambridge, MA: Harvard University Press https://www.hup.harvard.edu/books/9780674276604 (already called on in the chapter "The closed circle"), p. 43: "the presence of the exit alternative can therefore tend to atrophy the development of the art of voice"; p. 47: those who care most about quality, "the most active, reliable, and creative agents of voice", are the first to exit; p. 51: their departure "paralyzes voice by depriving it of its principal agents". ↩

16. Bødker, S., Ehn, P., Kammersgaard, J., Kyng, M. & Sundblad, Y. (1987). "A UTOPIAN Experience: On Design of Powerful Computer-Based Tools for Skilled Graphic Workers", in Bjerknes, G., Ehn, P. & Kyng, M. (eds.), *Computers and Democracy — A Scandinavian Challenge*. Aldershot: Avebury, pp. 251-278 https://search.worldcat.org/title/15489500. Retrospective: Sundblad, Y. (2011). "UTOPIA: Participatory Design from Scandinavia to the World". *History of Nordic Computing 3*, IFIP AICT 350. Berlin: Springer, pp. 176-186 https://doi.org/10.1007/978-3-642-23315-9_20. The project (1981-1986, Stockholm and Aarhus) paired the Nordic graphic workers' union (~120,000 members) with the design of page-layout and image-processing tools. The doctrinal summa: Ehn, P. (1988). *Work-Oriented Design of Computer Artifacts*. Stockholm: Arbetslivscentrum https://books.google.com/books/about/Work_oriented_Design_of_Computer_Artifac.html?id=dxCJswEACAAJ; the reference collection: Schuler, D. & Namioka, A. (eds.) (1993). *Participatory Design: Principles and Practices*. Hillsdale, NJ: Erlbaum https://www.routledge.com/Participatory-Design-Principles-and-Practices/Schuler-Namioka/p/book/9780805809510. ↩

30. The incompetent worker

The chapter "The invisible worker" showed one end of the chain: the competent contributor who carries the work unseen, to the point of exhaustion. Here is the other end, the most embarrassing to name: the contributor whose lack of skill can be neither said, nor corrected, nor set aside. The incompetence at issue is not first that of a person: it is that of an institution. The milieu has no way to handle competence — no way to filter it at the door, to assess it along the way, or to remedy its lack. The incompetent worker is not a fool to be mocked; he is the symptom of a house that knows neither whom it lets in, nor what its work is worth, nor how to tell someone to leave. Describing this void is drawing, in negative, the institution missing from every project recruited on enthusiasm alone.

Where does it come from, that a project teems with people who do not know how to do the work? From the fact that it is two things at once: a school and a product. People come to learn — for pleasure, for the challenge, for a skill to carry onto a résumé afterwards. Research has named these motives and measured them: the contributor signals his talent to future employers and peers[1]; he learns the trade by first participating from the margins, on the model of situated learning[2]; and motivation surveys place skill acquisition among the top reasons to contribute[3]. None of this is illegitimate — it is even precious. But a school and a product have opposite quality functions. In a

school, error is the method: you learn by getting it wrong. In a product a public depends on, error is the damage: you hunt it down and you hate it. Free software conflates the two, and runs the product like a school. Good for the apprentice, bad for the product: there is the confusion, stated.

An ordinary organization filters at the door — it hires —, evaluates along the way — it reviews the work — and can separate — it fires. Free software has none of the three: you do not choose your colleagues, no one has assessed their skills, and nothing allows parting with them. This is not an oversight, it is a recurring pattern. Elinor Ostrom, looking for what makes commons last, drew out design principles; three of them are exactly the ones missing here. Clear boundaries: who is a member, who holds rights. Monitoring: everyone's conduct and contribution are observed. Graduated sanctions: real consequences, up to exclusion[4]. Free software has neither boundary — benevolence makes every newcomer a member — nor monitoring — assessing a skill passes for an assault. As for the sanction, it exists, but inverted. Unable to dismiss someone frankly, people ignore him, or wear him down: silence on his contributions, the endless question, the reply that never comes. The milieu has replaced dismissal — honest, bounded, notified — with ostracism and attrition, which are none of those things. It is worse, and it is more hypocritical: no one tells a person he is not up to it; he is left to figure it out by knocking against the void. Where the durable common has a membership institution — frank about who enters, about what the work is worth, about who must leave — the project has only an open turnstile and a

hidden back door through which people are pushed out without a word.

For competence, here, cannot be said — even though it is defined without difficulty. To design, the chapter "The word and the thing" established it with Simon, is to turn an existing situation into a preferred one: to narrow the gap between what is and what we would want. Competence is then the property of an individual who narrows that gap, if only a little, without damaging what surrounds it. Both clauses count. The first meets the oldest psychological definition of competence — the ability to interact effectively with one's environment[5] — and what McClelland, as early as 1973, set against the cult of credentials: what attests competence is actual performance, never the title that presumes it[6]. The second clause is the one free software forgets: narrowing one gap by digging others is not competence, it is debt displacement. The one who fixes a bug by breaking three modules, who ships a feature by weighing down everyone's maintenance, who wins one user by repelling ten, has not brought the preferred situation closer: he has moved the gap elsewhere. And this environment not to be damaged by the act of design is not limited to the product to be shipped; it extends to the collective, to the social structure, to the rest of the living world and to ecosystems. It is the necessary guardrail against results at any cost. Competence is judged on a balance, not on a gesture. That balance is a verifiable fact: the quality of a fix, the same defect reintroduced three times, the lesson never learned in five years. But stating it is received as an offense of character — arrogance, brutality. It is the division of the chapter "The well-

tempered conflict" carried over onto persons: a judgment of competence is a judgment of truth, and a milieu without a metric of truth can only hear it as a matter of tone. There exists, then, no protocol by which "this decision was made by someone who was not in a position to judge it" can be said and admitted. Incompetence becomes unsayable; unsayable, it becomes unmanageable. You cannot correct what you are not allowed to observe.

And the taboo is all the more absurd in that incompetence is the easiest thing to establish — far easier than competence. Work well done is hard to prove: it is open-ended, cumulative, often invisible — the avoided bug leaves no trace, as the chapter "Counting the absent" showed. Harm done, for its part, shows: a regression, a broken module, a lost user are enough to signal it. Psychology has made a law of it — *bad is stronger than good*: more salient, more consistent, faster perceived than its positive counterpart[7]. Reliability engineering lives on it: you do not measure a system's "goodness", you count its failures, and safety long defined itself by the absence of accidents before anyone tried, with far more difficulty, to measure what goes well — the Safety-I / Safety-II division of the chapter "The root and the symptom". It is, finally, the very structure of falsifiability, the thread of this whole book: a competence cannot be proven, it would take an infinity of successful acts; it is refuted by a single harmful one. Competence is a pole never reached; incompetence, a fact that one case establishes. Characterizing it by the harm done is therefore not merely more convenient: it is the only

falsifiable grip — and that is why this chapter bears the name it bears.

A caveat at once, for measuring by harm alone has its flip side: whoever judges only on damage rewards the one who does nothing — the surest way to break nothing is to attempt nothing. That is why the measure remains a *balance*: harm is read in the debit column, but it is weighed against the good in the credit column. You do not set aside the clumsy one who takes useful risks and learns from his mistakes; you set aside the net harmful one — the one whose debit durably exceeds his credit, with no recovery in sight.

It remains to say how the sum is made, since a fault and a contribution are not in the same currency. There is no natural exchange rate between a broken module and a felicitous design; reducing both to a single figure — a "competence score" — would be the false precision the chapter "Measuring without lying to yourself" banished. Sociology has shown it: that reduction, *commensuration*, is never neutral — it is a social act that decides what counts and erases the rest[8]. So the balance is not computed in one stroke; it is read in two stages. First, floors: some harms disqualify whatever the contribution — the ones the definition has already ruled out, harm to users, to the collective, to the living world, the result obtained "at any cost". No brilliance redeems them; they are not weights on a scale, but values one refuses to put there — what decision psychology calls *protected* values[9]. Only above the floor do ordinary contributions and wrongs weigh against each other; and there, the relative weight of each is a choice of values, not a technical fact. It must therefore

be *declared*, with the rest of the values of the chapter "The pact", not improvised case by case according to who is being judged. The mandated judge applies it; falsifiability makes its application verifiable. No one will craft the perfect formula — part of the weighing remains a matter of judgment, and judgment does not automate. But a declared scale that can be contested is infinitely better than a secret balance held by the most powerful hand.

The price of incompetence is paid twice. First, training is work — real, unpaid, and dumped on the only ones who already know, that is, on the invisible worker already swamped: every newcomer to bring up to speed is a tax levied on the one least able to bear it. And that tax is never repaid: the trained apprentice often leaves once he has acquired what he came for. Know-how fits on a résumé, it travels; the maintenance of what he leaves behind does not — it stays, orphaned. The community trains at its own expense people who compound the fruit of that training elsewhere, and inherits the code they will no longer maintain. Second, know-how that is selected nowhere creates problems that fall on the competent, who, weary, end up leaving — the house empties (chapter "What insularity produces"), and what remains tightens into mediocracy (chapter "The closed circle"). Yet the picture must be turned around, for the newcomer is not only a cost. He sees what the veteran no longer sees: that the documentation is unreadable, that the welcome is hostile, that the interface is opaque to anyone discovering it. The "superfluous" that the old-timer disdains — a guide, a tutorial, a gentler entry — is often his most real contribution, and the meritocratic reflex that denies him a voice makes the project

blind to its own blind spots. The right answer is therefore not to close the door: a milieu that admits only ready-trained experts deprives itself of its only fresh eyes, and condemns itself to remain unreadable to anyone but itself. Training someone is, besides, care in the same sense as serving a user: work given to those who cannot yet, by those who already can. And like all care, it asks to be organized and paid for rather than quietly levied on the one who knows — failing which the house trains until its trainers are exhausted.

The second price of incompetence is paid when the training work has failed, or when training was never asked for — that is, when the lack of skill was not recognized and named, by its subject as much as by his supervisor: it is the transfer of badly done work onto colleagues. The one who does harm, displaces the problem or worsens it instead of solving it, or even creates problems through misjudgment or negligence, makes everyone carry the price of his errors. And in my experience of the free-software world, this incompetence is taboo: you can never say of someone that he is incompetent, whatever the weight of evidence bearing on the consequences of his repeated errors — it is straightaway an insult. The absence of pay, coupled with goodwill, seems to absolve absolutely every fault and every unlearned lesson, no matter how many people will suffer for it.

There remains the sore question: measure competence, fine, but how, and by whom? Free software already has its prestige metrics, and they are all superficial. Volume — the number of commits, seniority, presence on the channels — rewards activity, not its value: you can produce a great deal and improve nothing.

Tone — civility, conformity to the in-group — rewards agreeableness, not correctness: you can be charming and always wrong. Neither says whether the work is good. The only measure that does is falsifiable: the verifiable capacity to improve the project. And "improve" is neither vague nor merely technical — it is covering users' needs better, designing and documenting the tool so that it serves real people, building an infrastructure that holds, making the project's methods and social structure healthier. Competence is measured by what a contribution advances, on any one of these planes, against criteria set in advance. Such a criterion has a virtue that volume and tone lack: it is verifiable by a third party — anyone can take up the record and redo the count. But verifying is not deciding, and that is where the real questions begin.

First, what do you read competence on? On results — but objective measures of a piece of work and the judgments passed on it overlap only halfway: neither suffices alone[10], and any numeric indicator, as soon as it becomes the target, corrupts itself, as the chapter "Measuring without lying to yourself" established. On trajectory — the speed at which someone learns and stops repeating his errors, which the research on deliberate practice knows how to measure (chapter "The instrument and the machine"), but which requires a history and a baseline. On the judgment of peers — which catches what the number misses, but which the chapter "Thinking against oneself" showed to be biased by reputation and social signals. None of these grips is safe alone; they must be crossed. And even crossed, they do not contain

everything: part of competence is tacit, known without being sayable[11], and no scale exhausts it.

There remains the question I had dodged: a falsifiable criterion does not apply itself. Someone reads it, interprets it and decides — grants a right, opens a training path, pronounces an exclusion. That someone is a chief, and denying it would replay the illusion that the chapter "Freedom in a void" undid: the absence of structure does not abolish power, it hides it. There already is a chief, de facto: the merger — the one man who merges — whose veto goes uncontested. Falsifiability does not make him disappear; it makes his decisions contestable, and that is the whole difference. The de facto chief judges competence without criteria and without appeal — "shut up and fix it"; a mandated chief would judge it on criteria a third party verifies, by a decision that can be overturned by showing, evidence in hand, that it is wrong. On what basis to appoint that chief? Not on seniority or merge volume, but by the collective he affects, for a bounded, revocable mandate: democracy over who holds authority, the trade over how it is exercised — the assembly of the chapter "The pact". The power to judge competence is unavoidable; making it falsifiable, mandated and revocable is all that separates a chief from a tyrant. No one will ever build the perfect competence meter; we can make it so that being wrong there gets corrected, and being unjust there gets contested. It is little, but it is exactly what is missing.

Between exclusion by arrogance and limitless tolerance, there is what every durable institution has: a membership protocol, and a falsifiable one. Declare, at the door, what the project assumes you

know how to do — an explicit expectation, backed by the declared values of the chapter "The pact". Base prestige and membership on the measure of improvement above, not on volume or tone. Proportion authority to demonstrated competence, and let it grow with it: merge rights, weight in a decision are earned on the evidence and extend as the record thickens — the newcomer decides within a bounded perimeter, the expert within a wide one. Open a training path for whoever wants to learn, its cost owned and shared, not abandoned to a single person. Reserve exclusion — bounded, notified, open to appeal, Ostrom's graduated sanctions — for those who neither can nor will, and pronounce it frankly rather than through the void. It is not the punishment of an individual: it is the guarantee that the impact of one does not become the sentence of all. A contributor no one bounds does not just work badly in his corner; he pours his errors onto those who will have to repair them and onto the users who will suffer them. Setting someone aside, then, is protecting the collective from an externality, not chastising a person — and that is why it can, and must, be done without contempt. And above all, separate the learning area from production: let the beginner learn without his errors reaching the public that depends on the tool. Let the school be a school, in a space that is not the product. Nothing here requires cruelty; everything requires structure.

The incompetent worker was never the target. The target is a structure incapable both of growing know-how and of bounding its absence — and which, lacking both, ends up driving out those who know. The invisible worker met earlier and the incompetent

worker of this chapter are the two victims of one and the same missing institution: the first exhausts himself over what the second can neither learn nor leave. Naming the second without the first would be cruelty; naming them together is naming the void that produces them.

---

1. Lerner, J. & Tirole, J. (2002). "Some Simple Economics of Open Source". *Journal of Industrial Economics*, 50(2), 197-234 https://doi.org/10.1111/1467-6451.00174 — the *signaling incentive* (*career concern*): contributing to public, visible code signals one's competence to future employers and peers, a central economic motive for participating in free software. ↩

2. Lave, J. & Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation*. Cambridge: Cambridge University Press https://doi.org/10.1017/cbo9780511815355 — a trade is learned by first participating at the periphery of a community of practice; the model describes the typical entry into a free-software project, and reminds us that such a community is first of all a place of learning. ↩

3. On learning and skill acquisition (*human capital*) as dominant motives: Lakhani, K. R. & Wolf, R. G. (2005). "Why Hackers Do What They Do", in Feller, J. *et al.* (eds.), *Perspectives on Free and Open Source Software*. Cambridge, MA: MIT Press https://doi.org/10.7551/mitpress/5326.003.0005 ; Hars, A. & Ou, S. (2002). "Working for Free? Motivations for

Participating in Open-Source Projects". *International Journal of Electronic Commerce*, 6(3), 25-39 https://doi.org/10.1080/10864415.2002.11044241 . ↩

4. Ostrom, E. (1990). *Governing the Commons*. Cambridge: Cambridge University Press https://doi.org/10.1017/cbo9780511807763 — among the eight design principles of durable commons: (1) clearly defined boundaries (membership: who is a member); (4) monitoring of behavior; (5) graduated sanctions, up to exclusion. The chapter "The well-tempered conflict" already invoked the sixth (accessible conflict-resolution mechanisms). ↩

5. White, R. W. (1959). "Motivation Reconsidered: The Concept of Competence". *Psychological Review*, 66(5), 297-333 https://doi.org/10.1037/h0040934 — competence is defined there as the ability to interact effectively with one's environment, the source of a motivation of its own (*effectance*). ↩

6. McClelland, D. C. (1973). "Testing for Competence Rather Than for 'Intelligence'". *American Psychologist*, 28(1), 1-14 https://doi.org/10.1037/h0034092 — against the cult of tests and credentials: what attests competence is performance demonstrated in situation, not the title or the score meant to presume it. ↩

7. Baumeister, R. F., Bratslavsky, E., Finkenauer, C. & Vohs, K. D. (2001). "Bad is Stronger than Good". *Review of General Psychology*, 5(4), 323-370 https://doi.org/10.1037/1089-

2680.5.4.323 ; Rozin, P. & Royzman, E. B. (2001). “Negativity Bias, Negativity Dominance, and Contagion”. *Personality and Social Psychology Review*, 5(4), 296-320 https://doi.org/10.1207/s15327957pspr0504_2 — negative events have larger, more consistent and faster-perceived effects than positive ones of the same magnitude: the bad is detected better than the good. ↩

8. Espeland, W. N. & Stevens, M. L. (1998). “Commensuration as a Social Process”. *Annual Review of Sociology*, 24, 313-343 https://doi.org/10.1146/annurev.soc.24.1.313 — commensuration, the reduction of heterogeneous qualities to a common measure, is not a neutral operation: it is a social and political act that makes some things comparable and erases others. ↩

9. Baron, J. & Spranca, M. (1997). “Protected Values”. *Organizational Behavior and Human Decision Processes*, 70(1), 1-16 https://doi.org/10.1006/obhd.1997.2690 — values that people refuse to trade off against other goods, held to be non-negotiable whatever the gain offered in exchange. ↩

10. Bommer, W. H. *et al.* (1995). “On the Interchangeability of Objective and Subjective Measures of Employee Performance: A Meta-Analysis”. *Personnel Psychology*, 48(3), 587-605 https://doi.org/10.1111/j.1744-6570.1995.tb01772.x — objective measures of performance (the results) and subjective evaluations (the judgments passed on the same work) correlate only moderately: they are not interchangeable, and neither substitutes for the other. ↩

11. Polanyi, M. (1966). *The Tacit Dimension*. Garden City, NY: Doubleday — "we can know more than we can tell": part of know-how is *tacit*, mobilized without ever being fully made explicit or reduced to rules. ↩

31. The pact

This book has spent 5 parts taking things apart; it now owes its reader the reconstruction. All the flaws of the previous chapters share one root, and the chapter "Freedom in a void" named it: free software thought about freedom and never about the conditions of freedom — neither the user's capability, nor the worker's conditions, nor the structure of the collective. You do not repair that with more freedom; you repair it by supplying what freedom alone does not supply, and this chapter describes the instrument that would do it — an instrument not drawn for a single project: it is there for the taking, as is or amended, by any community that wants to last. The rest of the book rested on measurements; this one rests on a line of reasoning, consistent with what research knows about commons.

The economic diagnosis commands the rest. Free software makes two one-way transactions coexist, and it is their conjunction that poisons everything. The first: the user receives the product without giving anything back — not out of malice, but because 3 centuries of market have taught us that a dematerialized thing, infinitely downloadable, has no intrinsic value, hence no price. The second, symmetrical: the developer decides without accountability — the license having declared that he owed nothing to anyone, he owes nothing, not even an explanation. On one side a product taken without consideration, on the other a power exercised without responsibility: these are not two

problems, it is a single one, seen from its two ends. The milieu already has its pact, as it happens; you only have to listen to it being stated: "that's the implicit contract that comes with free software: You get the work of others to use and pay back with your time to report problems and help debugging them," wrote a Darktable core developer as early as 2016, before concluding that users who do not contribute are "a burden and not an asset"[1]. There is the existing contract, stated by one party: the user owes; the developer owes nothing — the license has exempted him. A contract in which only one party has obligations is not called a contract but a domination, and resolving it through a symmetrical domination (the developer bound to deliver whatever modifications the users demand) would not be enough to cancel it. The pact proposed here does nothing more than make the two arrows explicit, and reciprocal.

Now the moral responsibility all of this dodges must be laid down. When a hobby project becomes widespread enough for people to depend on it — users who have invested years of learning, entire libraries of edits, their workflow and sometimes their livelihood —, a responsibility is born: to make the project answer the needs of those who depend on it. I do say *a* responsibility, not *the* responsibility of the initial developer: nothing obliges the man who opened the worksite to carry it alone, and the chapter "The invisible worker" showed what demanding that of him costs. It is a responsibility of the *community* — the very one that currently allows itself a double discourse: on one side "free software frees you from the tech giants," on the other "the developers owe you nothing." The two

sentences cannot be true together: you do not recruit dependents in the name of freedom and then hold the volunteer's exemption up against them. Assuming this responsibility requires structures — that is this whole chapter — but first a change of outlook: seeing free software as a *public service*, in the full sense — a service rendered to a public that depends on it, with the continuity and the responsibility the word demands —, and not as a hive of individual initiatives that will survive, or not, according to their authors' mood and availability. The users' dependence is a fact, already acquired; the only question still open is whether it will be assumed by an institution, or exploited by the absence of one. The invisible worker of the chapter "The invisible worker" exhausts himself in the gap between the two, and the chapter "Three-tiered violence" showed what power without a counterpart produces, inside and out. Everything else — the insularity, the aggressiveness, the contempt for the user — grows on that soil, fed by a symmetrical frustration of each side toward the other.

A service is also judged by its fidelity to its declared mission. A project displays values: whom it designs for, what it refuses, what it promises. The display still has to be honest, and to stay honest. Three requirements follow, extending the falsifiability rule of this whole book. *Declare* those values explicitly, rather than leaving them as a welcome-page slogan. Make them *verifiable*: the gap between the professed value and the theory-in-use — whom you really serve, what you merge, what you refuse — must be measurable, as the chapter "Thinking against oneself" did against the records of a project. Make them *readable at the door*, finally: a

newcomer must be able to weigh the project's values against his own before committing years to it. Human resource management has measured this: a realistic job preview — given before the commitment, its servitudes as much as its attractions — reduces later departures, because it aligns expectations with reality[2]; and the congruence between a person's values and the organization's predicts satisfaction, commitment and retention[3]. The honest prospectus is therefore not a courtesy: it is an entry filter that spares costly defections. Without it, you discover the disagreement only once exit has become expensive — the uninformed entry is the hidden cause of the late departures of the chapter "The invisible worker". I have been through it: having joined on the faith of such a prospectus — "created for photographers, by photographers" —, I took years to measure the drift between the poster and the acts, and the finding was not a pleasant one. A project that declares its values falsely does not only deceive its users: it recruits its contributors on a prospectus it does not honor.

The way out is not to pick a side but to close the two arrows back into a circle: make the user contribute and the producer answer, not out of virtue, but because the structure binds them. That is exactly what a cooperative is — an old communism, proven and working, in which the enterprise, hence the project, is owned by those who use it and by those who work in it, sharing the voting power. The pact is there, and it is one of mutual responsibility: the producers answer for the users' needs; the users answer for the producers' working conditions — a safe, fair environment, and enough to live on. Neither the developer-dictator owed

eternal gratitude, nor the leech-user who demands much and gives little: two classes of members of the same house, each bound to the other. It is the exact opposite of the Wild West of the chapter "The closed circle": where there was no court, the cooperative institutes an assembly; where conflict was settled through access rights, it is settled through vote and mandate.

The division of legitimacies is the keystone, and it picks up the thread stretched from the chapter "Listening without obeying" to the chapter "The committee will invent nothing": the collective is sovereign over the *goals* and the *budgets*, the trade is sovereign over the *means*. Concretely, the general assembly secures an annual budget — most often through membership fees —, decides its use, and ranks the high-level priorities: improve masking, batch processing, the support of such-and-such format. It does not rule on the implementation, nor on the detail of the design, which lend themselves poorly to a vote and belong to the chapter "Stating the problem". There the developers will defend the deep workstreams — the overhaul of a technical debt, to reduce future maintenance cost; the users, the features they lack: the assembly deliberates and ranks, it does not execute. This division resolves at a stroke what the previous chapters set in opposition: it gives the user the power that was missing in the chapter "The closed circle" — the power to weigh on the direction, even to remove a team — without giving him the power to draw in the designer's place, which the chapter "Listening without obeying" denied him. Democracy over the ends, the trade over the means: the line is never crossed, in either direction.

An objection can come, this time, from the camp of care itself. The ethics of care distrusts contracts: Virginia Held showed that contractual thinking generalizes the case of independent equals who exist nowhere — the chapter "Freedom in a void" made use of it — and she was just as wary of a morality turning juridical[4]. A written pact, mandates, votes: is this not falling back into the same rut? This is where the object and the scale decide. Held was speaking of law colonizing care relations at the scale of a society; the pact above operates in a collective of a few dozen people where the alternative to written law has never been care — it has been the Wild West measured in the chapter "The closed circle". And her own conclusion settles it: "caring relations," she wrote, "should form the wider moral framework into which justice should be fitted." That is exactly the construction of this chapter — not replacing care with law, but giving care the few enforceable rules without which it remains a slogan. There remains the warning that history addresses to every care framework that succeeds. Virginia Henderson's definition of nursing and her grid of fundamental needs, taught to every generation of nurses, also ended up as dependency-assessment grids and billing flat rates — the instrument of emancipation become an accounting instrument[5]. A pact that ages well is not one that does not move: it is one whose assembly rereads the clauses asking whom they still serve. The day this pact serves the structure rather than the people, it will have to be revised in the spirit in which it was written — against attrition, not against people.

The user-member cooperative settles the problem that one-off donations have never managed to settle in free projects. This is

measured — the longitudinal studies of software crowdfunding show a real short-term effect and none in the long term, and the chapter "The invisible worker" said why: the donor is also the user, which sustains a consumer's relationship, not a co-owner's, and you do not fund research after the fact, just as you do not fund NASA once the rocket has left. The cooperative membership fee is of another nature than the donation: it does not buy a product already delivered, it acquires a share of the house and a voice in its choices; it is paid up front, because it funds a project you co-own, not a service or a product expected to prove itself before you commit. That is the decisive shift — from the grateful tip to the constitutive membership fee.

The pact also has a reach beyond comfort, and current events have made it literal. In 2024, a backdoor slipped into a library present on nearly every system in the world, not through a technical exploit but through the social engineering of a lone, exhausted maintainer, pushed for years to yield his place to his future attacker. The chapter "The invisible worker" drew a security requirement from it; the pact is its structural answer. An isolated maintainer, unpaid, without counter-power, is a vulnerability of the global supply chain; a worker held by a structure that pays him, surrounds him and shares the decisions is not one to the same degree. The worker's comfort, as was said, is not a nice-to-have; here it becomes a public-safety measure, and the pact, its instrument.

The fundamental objection remains: the proof is missing — no one can cite the great free-software cooperative that would have demonstrated the model. Three things temper this finding

without canceling it. First, the legal instrument already exists: the *société coopérative d'intérêt collectif*, in French law, is precisely a multi-stakeholder structure that binds beneficiary members and employee members — the pact does not need inventing, it needs applying. Next, the trajectory I propose is the one research observes: the projects that succeed are governed lightly and informally at small scale — one lead who does, a clear vision —, and give themselves formal institutions only as they grow; what this book seemed to set in opposition, the lone man today and the cooperative tomorrow, is one and the same road, taken at two moments — the one Elinor Ostrom brought to light by studying the commons that last, where formal institutions arrive only with scale[6]. Finally, the history of free software is full of collectives that ended up giving themselves the structure they denied — a constitution voted after a succession crisis, a steering council after a fiasco, a code of conduct imposed on even the most solitary of benevolent dictators[7]. "No structure" has never been a state, only a delay before the structure; the pact proposes not to wait for the crisis to write it.

The pact closes this part, and it closes more than the part. The book opened on a definition: to design is to solve a human's problem, which we have chosen to read technically. The pact draws the farthest consequence from it. The human in question is not only the user at the end of the chain: it is also the one who does the work, and it is the whole community that holds them together. The specification, the measurement, the engine, the simplicity, the instrument, the structure: everything this book has defended fits in a single idea, which the pact carries to its

collective form — behind every technical problem, there is a human. The pact gives the word "community," so easily brandished in the software world, its first meaning: a group of people who take care of one another, and who commit to one another — not just an interest group excited about the same things.

It remains to look this idea in the face. The whole discipline of this book has a name that engineering never utters; the conclusion exists to utter it.

---

1. https://discuss.pixls.us/t/975 (March 2016, archived: see chapter "What insularity produces"): "that's the implicit contract that comes with free software: You get the work of others to use and pay back with your time to report problems and help debugging them. Freeloaders are a burden and not an asset". ↩

2. Phillips, J. M. (1998). "Effects of Realistic Job Previews on Multiple Organizational Outcomes: A Meta-Analysis". *Academy of Management Journal*, 41(6), 673-690 https://doi.org/10.2307/256964 — the *realistic job preview* (showing the candidate the constraints as much as the advantages, before hiring) lowers staff turnover and brings expectations closer to the lived reality. ↩

3. Kristof-Brown, A. L., Zimmerman, R. D. & Johnson, E. C. (2005). "Consequences of Individuals' Fit at Work: A Meta-Analysis of Person-Job, Person-Organization, Person-Group,

and Person-Supervisor Fit". *Personnel Psychology*, 58(2), 281-342 https://doi.org/10.1111/j.1744-6570.2005.00672.x — person-organization fit, of which value congruence is the core, predicts job satisfaction, commitment and retention. ↩

4. Held, V. (2006). *The Ethics of Care: Personal, Political, and Global*. New York: Oxford University Press — on contractual generalization, p. 81; on the place of justice: "caring relations should form the wider moral framework into which justice should be fitted", p. 71 https://doi.org/10.1093/0195180992.001.0001 . ↩

5. Virginia Henderson's model (the 14 fundamental needs grid) structures the data collection of French nursing; the official course material of the IFSI (the French nursing schools) teaches that this model "influenced [...] the financial coverage of health care" (our translation), AGGIR grid in evidence — the loss-of-autonomy assessment that conditions the dependency allowance (course material, IFSI de Troyes https://www.ifsi-troyes.fr/sites/default/files/le_modele_conceptuel_decrit_par_virginia_henderson_2.pdf); the Bilan de Soins Infirmiers, built on the same needs logic, today prices home nursing care in daily flat rates. ↩

6. On the emergence of legitimacy: O'Mahony, S. & Ferraro, F. (2007). "The Emergence of Governance in an Open Source Community". *Academy of Management Journal*, 50(5), 1079-1106 https://doi.org/10.5465/amj.2007.27169153 (a longitudinal study of Debian). On the light-then-formal trajectory: Schweik, C. M. & English, R. C. (2012). *Internet*

Success. Cambridge, MA: MIT Press https://doi.org/10.7551/mitpress/9780262017251.001.0001 . The commons framework, as formulated in Ostrom, E. (1990). *Governing the Commons*. Cambridge: Cambridge University Press https://doi.org/10.1017/CBO9780511807763 — whose 6th design principle is the existence of accessible, low-cost conflict-resolution mechanisms. The commons framework adapted to software (where the risk is not Ostrom's overexploitation but under-provision and burnout): Frischmann, B. M., Madison, M. J. & Strandburg, K. J. (eds.) (2014). *Governing Knowledge Commons*. Oxford: Oxford University Press https://doi.org/10.1093/acprof:oso/9780199972036.001.0001 ; upstream, Benkler, Y. (2002). "Coase's Penguin, or, Linux and *The Nature of the Firm*". *Yale Law Journal*, 112(3) https://doi.org/10.2307/1562247 . The French legal framework: société coopérative d'intérêt collectif, loi n° 2001-624 du 17 juillet 2001 https://www.legifrance.gouv.fr/jorf/id/JORFTEXT000000757800 . ↩

7. Public examples, cited for the role of the institutions and not of the people: the adoption by a major programming language of an elected steering council after the resignation of its historic benevolent dictator (2018); the voted constitution of a major community operating system (1998), with an elected leader and general resolutions as a guardrail; the institution of a governance council in a young systems language after a crisis (2023), whose post-mortem concedes that the deciding body "lacked clear rules and processes for

decision making and communication"; the adoption of a code of conduct by the most centralized of kernels (2018). Proclaimed "meritocracy" and actual "pushyocracy": Nafus, D. (2012). "'Patches don't have gender'". *New Media & Society*, 14(4) https://doi.org/10.1177/1461444811422887 . ↩

Conclusion — Taking care

The technical part of this book can be summed up like this: design is a discipline we impose on ourselves everywhere our natural inclination — laziness — tends to make us favor, in the short term, our own interests as designers: the quick solution, barely defined, that does not listen to the user's needs (whether it ignores them out of condescension or lets itself be blinded by "what the user wants" taken literally), that stops too early in the climb back up the problem's causes, that looks under the streetlight. Technology is only a chosen lens on it, not the goal.

The social part can be summed up by the fact that no technical object is ever neutral or exists outside the group that creates it, and that the question is not whether it belongs to its designers or to its users: it belongs to everyone, and it must be able to meet the needs of the ones without exhausting the others in maintenance. Thus the design of the social structure that produces the technical artifact is part of the design of the artifact itself, at the level of governance as much as at the level of the organization of work. Here, the discourses that legitimize the oppression of the user must be refused just as firmly as those that try to build a Christ-like figure of the designer, one who ignores his own needs, his safety and his comfort.

The part specific to free software buries for good the false beliefs the free-software crowd holds about itself: they emancipate only

the privileged, and their supposedly horizontal social structures are dictatorships that strip themselves of the very ability to think their own mechanisms of domination, while sliding into sectarian and dogmatic behavior that silences criticism from the inside as much as criticism from the outside.

The part "Freedom and its conditions" closed on a pact; this conclusion pronounces head-on what the prologue announced and what the pact, like the whole book before it, kept seeking, gesture after gesture. Listen before deciding, characterize the need before the solution, test the reception instead of presuming it, maintain instead of dumping, answer for what you ship: chapter after chapter, I have described a discipline. Its real name does not belong to the vocabulary of engineering, and it is time to look it in the face: taking care.

The word jars. Design as we practice it comes from engineering, engineering comes from the machine, and the machine comes from a world of men. This is not a figure of speech: the history of technology has documented the deliberate construction of the engineer as a masculine figure — at the turn of the twentieth century, the trade defined itself by excluding, and the exclusion is part of the definition[1]. A century later, the imprint is intact. In the United States, women earn a quarter of engineering degrees; mechanical engineering, the largest discipline by volume and the historical matrix of industrial design methods, sits at 18% — and electrical engineering and computer science do worse[2]. Free software, which tells its own story as the open utopia of that world, is its caricature: 1.5% women among developers in the early 2000s, 3% in the big 2017 survey, and, depending on the

measurement method — self-reported survey or inference over commit records — between 8 and 14% today. Never, in 20 years, has the order of magnitude passed one seventh[3]. The milieu that writes everyone's tools is, socially, the narrowest of all the technical trades. This is not a moral remark: it is a design datum. A homogeneous milieu designs for itself; the part "Freedom and its conditions" spent 4 chapters measuring what that produces, and the present chapter says what is missing across from it.

What is missing has a name in moral philosophy, and it so happens that the name was forged by the very women engineering had excluded. The ethics of care was born in the early 1980s with Carol Gilligan, who showed that alongside the morality of principles and rights there exists a morality of relation and concrete responsibility, first heard as "a different voice"[4]; with Nel Noddings, who described *engrossment* — attention absorbed by the other — and *motivational displacement* toward the other's needs: what this book has called listening to the testimony rather than the prescription[5]; then with Joan Tronto, who tore care away from the register of feminine virtues to make it a political category. Virginia Held drew the complete moral theory from it, and her theorem is the one this book reaches by measurement: "There can be care without justice [...] There can be no justice without care, however, for without care no child would survive and there would be no persons to respect"[6]. Tronto and Berenice Fisher's definition describes our trade exactly: care is "a species activity that includes everything that we do to maintain, continue, and repair 'our world' so that we can live in it as well as possible"[7]. To maintain, continue and

repair a world so that we can live in it: what is design — software design in particular — if not that? Tronto breaks care down into phases — caring about, taking care of, care-giving, care-receiving, and later caring *with* — and the correspondence with this book is term for term: caring about is stating the problem (the chapter "Stating the problem"); taking care of is specifying and committing (the chapters "Measuring without lying to yourself" and "Ship, don't dump"); care-giving is building; care-receiving is testing the real reception instead of presuming it (the chapter "Counting the absent") — the phase engineering always forgets, because it presupposes that the recipient of the care has a say; and caring *with* is the governance the chapters "Enabling without holding the brush" and "The pact" sketched[8]. Design is an activity of care. It was invented, historically, by the one milieu that refused to think of it as such — and research eventually made the connection: the ethics of care was brought, late, into science and technology, and all the way into human-computer interaction, where a feminist design agenda has existed in black and white since 2010[9]. Yet even the design literature admits the word "care" only where it touches the hospital: the one general-audience book that dares "Design as Caregiving" is a book about health systems[10]. Care, apparently, is for the sick; the well are entitled to nothing but features.

It is precisely the reverse, and here is the heart of my thesis. The world is being technicized, and it is being technicized without asking the opinion of those who undergo it. Ellul said it 70 years ago: technology augments itself, imposes itself as an environment, and never goes to a vote[11]. Langdon Winner said

it in an image: “we so willingly sleepwalk through the process of reconstituting the conditions of human existence”[12]. Technologies are pushed from the top down, by those who know and who can, onto those who need and who endure. Nobody voted for the online-only counter, for the mandatory app, for the end of the paper form; it was decreed progress, and “you can’t stop progress” is not an observation: it is a permit. A permit to bypass the democratic discussion about which technologies we want, which ones we accept, and what consequences we tolerate them having on our lives — the historian David Noble showed what that rhetoric of the inevitable covers in very avoidable choices, made by very few people[13]. And when the philosophy of technology looks for the antidote, it finds it on the side of users intervening in design itself: what Andrew Feenberg calls democratic rationalization — laypeople forcing technology to integrate their interests against the technical hierarchy[14]. That is, in philosopher’s language, exactly the program of the chapter “Enabling without holding the brush” and of the pact.

In a world artificialized by force in this way, digital illiteracy is not an individual weakness: it is a wound inflicted by the environment. Research on the digital divide has been repeating it for 20 years: access is not enough, skills are socially distributed, and non-use is a fact of structure, not a defect of character[15]. Norman documented the intimate mechanism: faced with a badly designed tool, the user does not blame the tool, he blames himself, all the way to taught helplessness — he stops trying[16]. That is the state the recipient is in. And what does free software, which presents itself as his liberation, answer him? Contempt.

There, the digital illiterate is treated as a burden; the newcomer runs into documented social barriers — hostile answers, condescension, jargon-as-password — that research has cataloged right down to the vocabulary of the mailing lists[17]; the culture of "just read the manual" stands in for a welcome, and the part "Freedom and its conditions" showed the club this builds. The chapter "Three-tiered violence" gave the verdict on that contempt: when "human error" is almost always a design error, despising the failing user is despising your own badly done work. The one who needs care most is treated as the one who deserves it least.

This is where the medical analogy stops being an image. Medicine helps living beings live: it reduces suffering, lightens ills and impediments, restores movement and comfort — the relief of suffering and the cure of disease, "twin obligations" of the profession, says a classic text[18]. Design helps *technicized* beings live in an artificialized world: it reduces friction, lightens procedures, restores the capacity to act of the one whom technology had reduced to helplessness. Bernard Stiegler gave this kinship its concept: technology is a *pharmakon*, at once "what makes it possible to take care and what must be taken care of — a curative power in the measure and the excess in which it is a destructive power"[19]. The designer is the pharmacist of this poison-remedy: the same technical object that emancipates, badly dosed, badly thought through, destroys — and there is nobody but us between the dose and the patient.

If design is the medicine of the artificial world, then it needs what medicine has: an oath, and a professional ethics. The idea is

not new. Hippocrates first, in the actual text rather than in the legend — the maxim is not in the Oath, it is in the Epidemics: in disease, keep two things in view, "to help, or at least to do no harm"[20]. Victor Papanek next, who in 1971 opened his book on the designer's social responsibility with the most insolent sentence in the discipline: "There are professions more harmful than industrial design, but only a very few of them"[21]. Mike Monteiro, closer to us, demands for designers a code of ethics and a professional accountability organized on the model of medicine — the phrase "designer's Hippocratic oath" is his commentators', but the argument is very much his[22]. Free software itself saw the birth of a "Hippocratic License", forbidding the use of a program to violate human rights — and the open-source establishment rejected it in the name of the canonical definition, which protects the "freedom zero" to do harm[23]: down to its very licenses, the milieu has chosen the abstention from care. Here then, to finish, is the oath I formulate, and it condenses the book. *To help, or at least to do no harm*: not to destroy the capital of the one who trusted me — his data, his years of learning, his workflow (the chapter "Ship, don't dump"). Not to make those who asked for nothing carry the cost of my comfort as a developer (the chapter "Working for the work"). To design for the one who will not be able to defend himself against my design: the illiterate, the rushed, the tired — the user in a state of weakness is the nominal case, not the edge case (the chapter "Three-tiered violence"). To listen to the testimony before deciding, and to test the reception after deciding, because care that does not check whether it was received is not care, it is a broadcast (the chapters "Stating the problem" and "Counting the absent"). To document what my tool

does *to* its user, not only what it does *for* him. To keep my choices reversible as long as I have no proof that they do care (the chapter "Listening without obeying"). And to answer for the whole before those who depend on it, in a structure where they have a voice — for care without accountability becomes power and oppression again (the chapter "The pact").

The ethics of care addresses this warning to itself, for that matter: care is nothing innocent — it can oppress, moralize, decide another's good in his place — and María Puig de la Bellacasa made this a condition of the concept rather than an embarrassment[9]. This oath escapes the objection in only one way: it binds no one but the one who swears it, and each of its clauses can be checked. I can be caught failing it; that is the point.

The two objections remain to be faced, and I take the strongest version of each. The first comes from inside the ethics of care itself: by speaking of care, are we not renewing the old division — solicitude for the women, technology for the men — that is, the very essentialism these women fought? The objection was made to Gilligan and to Noddings, and the answer is Tronto's: care is not a feminine disposition, it is work — universally required, historically assigned to subordinates, and devalued for that very reason[8]. Demanding that engineering learn care is not feminizing a bastion: it is demanding that a trade of the privileged take back its share of a work it has always delegated — to the women who write the documentation, to the support teams, to the peer-help forums, to all the invisible trades of the chapter "The invisible worker" — and the sociology of computing shows who, statistically, ends up doing it. The second objection is

the engineer's: is care not paternalism — simplifying by force, deciding in the user's place, infantilizing him? It deserves better than a shrug, for a care that confiscates is indeed one more domination, and the chapter "Enabling without holding the brush" showed "simplifications" that were amputations. But the objection confuses care with assistance. The goal of care, in medicine as in design, is the autonomy of the one cared for, and Illich wrote the criterion once and for all: a society "in which modern technologies serve politically interrelated individuals rather than managers" is what he called convivial, because "people need new tools to work with rather than tools that 'work' for them"[24]. The caring profession itself had set the criterion before him: the founding definition of modern nursing — Virginia Henderson, 1966, learned by every nursing student in France — assists the individual, "sick or well", in "the performance of those activities contributing to health or its recovery (or to peaceful death) that he would perform unaided if he had the necessary strength, will or knowledge", and does so, says the decisive clause, "in such a way as to help him gain independence as rapidly as possible"[25]. A care that measures itself by the speed of its own disappearance cannot be the paternalism it stands accused of. Taking care of the user is not holding the brush in his place; it is making sure the tool fits his hand, sits within his reach, and stays under his control. Paternalism decides for the other; care makes the other capable of deciding. The whole difference of this book lies in that fold.

One last word remains to be disarmed, and it is addressed to me. Once again this week, I was thanked for my "vision" — the word

is in the message, and the fault is first mine: the description of my own fork displays it, "a Darktable fork minus the bloat plus some design vision", written on a day when characters were counted[26]. The word must be refused all the same, and the refusal sums up the book. "Vision" is the hero narrative under its other name: a rare gift, lodged in an exceptional individual, innate therefore untransmittable, admirable therefore beyond discussion — the exact register the chapter "Non-heroic ambition" dismantled in maintainers, and which will not be reinstalled for the designer's benefit. What people call my vision, this book has just gone through, piece by piece: it is a method. Part of it came to me from engineering school, part from intuition corrected along the way; and everything in it that was worth anything let itself be reduced to things research studies, to questions you force yourself to ask in the right order, to priorities you forbid yourself to betray — and, at the end, to the care of those who need to be cared for. Medicine does not say of a good clinician that he has a vision; it says that he has a discipline. Henderson defined nursing without one word for talent[25]. There are no visionaries in design: there are people who imposed a discipline of care on themselves — and collectives that made it possible for them, or impossible. What passes for a gift can be transmitted; what can be transmitted can be taught; and what can be taught excuses no one anymore.

I can now say why the book had to end here. The question that opened it was technical: what is designing? The answer, at the end of the road, no longer is. To design is to care for others through a technical object — it is the form care takes when it has

to pass through the machine. A world that is technicized without anyone being asked manufactures, every day, technicized beings who did not choose to be and who must live anyway. Someone has to help them live there, as medicine helps people live, period. That someone is us. This trade is not a right to be exercised over others; it is a burden to be carried for them. Free software spent 30 years liberating code. The thing left to liberate is the user — and you do not liberate someone you despise.

---

1. Oldenziel, R. (1999). *Making Technology Masculine: Men, Women, and Modern Machines in America, 1870-1945*. Amsterdam: Amsterdam University Press. https://doi.org/10.5117/9789053563816 Scope of the study: the United States, 1870-1945 — the generalization to the whole trade is my extrapolation, which the contemporary figures of the following note support. ↩

2. American Society for Engineering Education (2024). *Engineering & Engineering Technology by the Numbers, 2023*. Washington, DC: ASEE https://ira.asee.org/by-the-numbers/ , tables 1 and 4 — women: 24.6% of engineering bachelor's degrees; mechanical engineering: the largest discipline by volume (29,792 degrees) and 17.9% women; aerospace (17.4%), petroleum (16.4%), computer science (14.9%) and electrical engineering (14.6%) do worse still. American data; the proportions vary by a point or two from one edition to the next. ↩

3. 1.5%: FLOSS developer survey 2002 (Ghosh, R. A., Glott, R., Krieger, B. & Robles, G., *Free/Libre and Open Source Software: Survey and Study*, International Institute of Infonomics, Univ. of Maastricht, 2002) https://www.math.unipd.it/~bellio/FLOSS%20Final%20Report%20-%20Part%204%20-%20Survey%20of%20Developers.pdf, a figure taken up and discussed — against 28% in proprietary software (NSF, 2004) — by the gender report of the FLOSSPOLS project: Nafus, D., Leach, J. & Krieger, B. (2006). "Gender: Integrated Report of Findings", deliverable D16, Univ. of Cambridge / European Commission, p. 4. https://www.jamesleach.net/downloads/FLOSSPOLS-D16-Gender_Integrated_Report_of_Findings.pdf 3%: GitHub, *Open Source Survey 2017* https://opensourcesurvey.org/2017/ ("95% of respondents are men; just 3% are women and 1% are non-binary"). 8.1% of public commits (13.5% of authors whose gender can be inferred): Rossi, D. & Zacchiroli, S. (2022). "Worldwide Gender Differences in Public Code Contributions…", *ICSE-SEIS'22*, ACM https://doi.org/10.1145/3510458.3513011 — inference by first name, covering 64% of authors, with cultural biases acknowledged by the authors. 14%: Linux Foundation Research (2021), *Diversity, Equity, and Inclusion in Open Source* https://www.linuxfoundation.org/research/the-2021-linux-foundation-report-on-diversity-equity-and-inclusion-in-open-source — a self-reported survey. No single figure is honest without its method; the range, though, is stable in its ugliness. ↩

4. Gilligan, C. (1982). *In a Different Voice: Psychological Theory and Women's Development*. Cambridge, MA: Harvard University Press. https://doi.org/10.2307/j.ctvjk2wr9 ↩

5. Noddings, N. (1984). *Caring: A Feminine Approach to Ethics and Moral Education*. Berkeley: University of California Press https://books.google.com/books/about/Caring.html?id=vkMkLi6pnMYC — the two states she describes are *engrossment* (attention absorbed by the other) and *motivational displacement* (the shift of motivation toward the other's needs). ↩

6. Held, V. (2006). *The Ethics of Care: Personal, Political, and Global*. New York: Oxford University Press, p. 17. https://doi.org/10.1093/0195180992.001.0001 On the same page, Held (1929-2026) held care to be "probably the most deeply fundamental value". ↩

7. Fisher, B. & Tronto, J. C. (1990). "Toward a Feminist Theory of Caring", in E. K. Abel & M. K. Nelson (eds.), *Circles of Care*. Albany: SUNY Press, p. 40 https://sunypress.edu/Books/C/Circles-of-Care ; taken up in Tronto, J. C. (1993). *Moral Boundaries: A Political Argument for an Ethic of Care*. New York: Routledge, p. 103. https://doi.org/10.4324/9781003070672 ↩

8. Tronto, *Moral Boundaries*, *op. cit.*, pp. 105-108 (the four phases, and their moral qualities: attentiveness, responsibility, competence, responsiveness); the fifth phase, *caring with*, is added in Tronto, J. C. (2013). *Caring*

Democracy: Markets, Equality, and Justice. New York: NYU Press https://doi.org/10.18574/nyu/9780814770450.001.0001 — which also carries the anti-essentialist argument: care as devalued and assigned work, not as a gender disposition. ↩ ↩

9. Puig de la Bellacasa, M. (2011). "Matters of care in technoscience: Assembling neglected things". *Social Studies of Science*, 41(1), 85-106 https://doi.org/10.1177/0306312710380301 ; and (2017) *Matters of Care: Speculative Ethics in More Than Human Worlds*. Minneapolis: University of Minnesota Press. https://doi.org/10.5749/9781452953465 The HCI agenda: Bardzell, S. (2010). "Feminist HCI: Taking Stock and Outlining an Agenda for Design". *CHI '10*, ACM, pp. 1301-1310 https://doi.org/10.1145/1753326.1753521 — six qualities: pluralism, participation, advocacy, ecology, embodiment, self-disclosure. ↩ ↩

10. Jones, P. H. (2013). *Design for Care: Innovating Healthcare Experience*. Brooklyn, NY: Rosenfeld Media https://rosenfeldmedia.com/books/design-for-care/ , ch. 1, "Design as Caregiving" — the frame is explicitly that of health systems; the extension of care to all design is my thesis, not his. ↩

11. Ellul, J. (1954). *La Technique ou l'Enjeu du siècle*. Paris: Armand Colin https://www.sudoc.fr/013436104 — the self-augmentation and the autonomy of technology, which imposes itself as an environment without deliberation, are the heart of the book; stated here in indirect speech. ↩

12. Winner, L. (1986). *The Whale and the Reactor: A Search for Limits in an Age of High Technology*. Chicago: University of Chicago Press https://doi.org/10.7208/chicago/9780226902098.001.0001, ch. 1, p. 10: "we so willingly sleepwalk through the process of reconstituting the conditions of human existence" — "technological somnambulism". ↩

13. Noble, D. F. (1995). *Progress Without People: New Technology, Unemployment, and the Message of Resistance*. Toronto: Between the Lines https://archive.org/details/progresswithoutpoooonobl (1st ed.: *In Defense of Luddism*, Chicago: Charles H. Kerr, 1993) — in indirect speech. ↩

14. Feenberg, A. (1999). *Questioning Technology*. London: Routledge. https://doi.org/10.4324/9780203022313 The concept first appears under the name "subversive rationalization": Feenberg, A. (1992). "Subversive Rationalization: Technology, Power, and Democracy". *Inquiry*, 35(3-4), 301-322 https://doi.org/10.1080/00201749208602296; it is renamed "democratic rationalization" in the 1999 book. ↩

15. Van Dijk, J. (2005). *The Deepening Divide: Inequality in the Information Society*. Thousand Oaks: Sage https://doi.org/10.4135/9781452229812 — access as a chain of barriers (motivation, equipment, skills, uses); Hargittai, E. (2002). "Second-Level Digital Divide: Differences in People's Online Skills". *First Monday*, 7(4)

https://doi.org/10.5210/fm.v7i4.942 ; Wyatt, S. (2003). "Non-Users Also Matter", in N. Oudshoorn & T. Pinch (eds.), *How Users Matter*. Cambridge, MA: MIT Press, pp. 67-79 https://doi.org/10.7551/mitpress/3592.003.0006 — against reading non-use as an individual deficit. ↩

16. Norman, *The Design of Everyday Things*, *op. cit.*, ch. 2, sections "Falsely Blaming Yourself" and "Learned Helplessness" (1988 ed., pp. 39-40): the user's self-blame and taught helplessness. ↩

17. Steinmacher, I., Conte, T., Gerosa, M. A. & Redmiles, D. (2015). "Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects". *CSCW '15*, ACM, pp. 1379-1392 https://doi.org/10.1145/2675133.2675215 — 58 barriers listed, thirteen of them social (hostile welcome, contemptuous or absent answers); as a complement on the verbal register of the milieu: Squire, M. & Gazda, R. (2015). "FLOSS as a Source for Profanity and Insults". *HICSS-48*, IEEE, pp. 5290-5298. https://doi.org/10.1109/hicss.2015.623 The phrase "just read the manual" is my polite paraphrase of an acronym these studies do not use. ↩

18. Cassell, E. J. (1982). "The Nature of Suffering and the Goals of Medicine". *New England Journal of Medicine*, 306(11), p. 639 https://doi.org/10.1056/NEJM198203183061104 : "The relief of suffering and the cure of disease must be seen as twin obligations of a medical profession that is truly dedicated to the care of the sick"; developed in Cassell, E. J. (1991). *The Nature of Suffering and the Goals of Medicine*.

New York: Oxford University Press. https://doi.org/10.1093/acprof:oso/9780195156164.001.0001 ↩

19. Stiegler, B. (2008). *Prendre soin. Tome 1 : De la jeunesse et des générations*. Paris: Flammarion https://www.cultura.com/p-prendre-soin-de-la-jeunesse-et-des-generations-9782081207363.html ; and (2010) *Ce qui fait que la vie vaut la peine d'être vécue. De la pharmacologie*. Paris: Flammarion https://editions.flammarion.com/ce-qui-fait-que-la-vie-vaut-la-peine-detre-vecue/9782081220355 — the quotation comes from the publisher's presentation of the 2010 volume, checked against the copy: « ce qui permet de prendre soin et ce dont il faut prendre soin — puissance curative dans la mesure et la démesure où c'est une puissance destructrice » (our translation). The title of the present chapter owns the echo. ↩

20. Hippocrates, *Epidemics*, book I, 2nd section, § 5 — quoted here in the standard English rendering ("make a habit of two things: to help, or at least to do no harm"); French text in É. Littré's translation, *Œuvres complètes d'Hippocrate*, vol. II, Paris: J.-B. Baillière, 1839-1861. https://archive.org/details/BIUSante_34859x02 Contrary to folklore, the Latin formula *primum non nocere* does not appear in the Oath; on its actual origin, see Smith, C. M. (2005). "Origin and Uses of Primum Non Nocere". *The Journal of Clinical Pharmacology*, 45(4), 371-377. https://doi.org/10.1177/0091270004273680 ↩

21. Papanek, V. (1971). *Design for the Real World: Human Ecology and Social Change*. New York: Pantheon Books https://books.google.com/books/about/Design_for_the_Real_World.html?id=gf5TAAAAMAAJ — the opening sentence, quoted here: "There are professions more harmful than industrial design, but only a very few of them" (preface; pagination varies from edition to edition). ↩

22. Monteiro, M. (2019). *Ruined by Design: How Designers Destroyed the World, and What We Can Do to Fix It*. San Francisco: Mule Books https://www.ruinedby.design/ ; and (2017) "A Designer's Code of Ethics", Dear Design Student https://deardesignstudent.com/a-designers-code-of-ethics-f4a88aca9e95 . The attribution is to be taken to the letter: Monteiro argues for a code of ethics and a professional accountability organized on the medical model; the phrase "designer's Hippocratic oath" comes from his commentators, not from his texts. ↩

23. Ehmke, C. A. (2019). *The Hippocratic License* https://firstdonoharm.dev/ (later versions: HL3, Organization for Ethical Source). The rejection rests on the *Open Source Definition* (criteria 5 and 6: no discrimination against persons or against fields of endeavor, OSI) https://opensource.org/osd and, on the FSF side, on "freedom 0", to run the program for any purpose. https://www.gnu.org/philosophy/free-sw.html ↩

24. Illich, I. (1973). *Tools for Conviviality*. New York: Harper & Row https://openlibrary.org/works/OL2848898W — "Such a

society, in which modern technologies serve politically interrelated individuals rather than managers, I will call ‘convivial’” (introduction); “People need new tools to work with rather than tools that ‘work’ for them” (ch. II, “Convivial Reconstruction”). The French version, *La convivialité*, Paris: Seuil, 1973, was established by the author with L. Giard and V. Bardet https://catalogue.bnf.fr/ark:/12148/cb43897048t and states the same criterion in words of its own: « j’appelle société conviviale une société où l’outil moderne est au service de la personne intégrée à la collectivité, et non au service d’un corps de spécialistes. Conviviale est la société où l’homme contrôle l’outil » (introduction). ↩

25. Henderson, V. (1966). *The Nature of Nursing*. New York: Macmillan — the definition is quoted here in her own English; French translation *La nature des soins infirmiers*, Paris: InterÉditions, 1994. The definition, first published in 1955 (Harmer & Henderson, *Textbook of the Principles and Practice of Nursing*, 5th ed., Macmillan), was spread worldwide by the International Council of Nurses (*Basic Principles of Nursing Care*, Geneva, 1960, translated into some thirty languages); in France, Henderson’s grid of 14 fundamental needs still structures nursing data collection. ↩ ↩

26. Message received on July 31, 2026 on the Ansel repository: “Thanks to Aurelien for your vision, patience and perseverance, and to Jiyone for taking over and keeping the project going!” (discussion #812

https://github.com/aurelienpierreeng/ansel/discussions/812#discussioncomment-17853594). The repository description, written by me: “A Darktable fork minus the bloat plus some design vision.” ↩

Epilogue — Behind every problem, a human

It took the writing of this book to put words and concepts, defined outside of me, on the fiasco of my experience with Darktable. The conclusion holds in few words: irreconcilable goals, values and priorities. And I will add bluntly that my goals and my priorities are better than theirs, as measured by social impact outside the closed circle. This is not a matter of persons. My priorities are measured against those who are not in the room: the user on whom digital technology was imposed without being asked, and who has no way to defend himself against a bad tool. Theirs are measured against themselves — the pleasure of coding, the techniques you get to learn, the insularity you get to keep. Serving those who cannot defend themselves rather than yourself is not a virtue I claim for myself: it is the definition of a social benefit. And mine can be verified — my user surveys are published with their data — while theirs invokes an "emancipation" that the record refutes. A goal you measure and own is worth more than a goal that hides behind a slogan.

We delegate to the high-school computer club the design and manufacture of tools meant for people who are not in the room: people who have fun with computers, want to learn new techniques, and are looking for an alibi to do so. This amateurism is structural: it is how contributors are recruited, exonerated of

all responsibility by the free license and the volunteer nature of their work. It is neither good nor bad, just out of place if the aim is to deliver a social benefit in a context where, I repeat, digital technologies were imposed on people without asking them anything, on a very short timescale.

What is culpable is the double discourse — the one the prologue announced, and that the chapters have since measured against the exhibits: the façade that claims user emancipation, unconditional inclusivity, tools mature enough for professional use, the "for photographers, by photographers" I let myself be taken in by, while the reality shows overgrown children playing with code, in contempt of users, and with an absence of rigor and method they would probably not dare allow themselves in their "real job" (that is a charitable hypothesis on my part; in truth I have no idea).

This way of creating free software — with no design method, no structure to guarantee one, no accountability attached to named and bounded responsibilities, with a fake horizontality that conceals the real power of a few, applying value scales and cargo cults that the scientific literature dismantles — serves no one: it creates nothing but entropy. People bustle, people get excited, but past work only increases the bill for future work, a bill someone else will pay, and users are the great forgotten of the system. Instead, people prefer to create yet another new image editor — now entirely written by AI thanks to LLMs, and therefore within almost anyone's reach — and pile up half-finished products. You need only pick yours — an illusion of choice: a real choice would

be between tools that work; here, you only choose which features you will do without.

This system is blind to itself: the absence of what it regards as dirty words (method, structure, responsibilities, organization, falsifiable goals and metrics, competence) is what costs it the most in time and labor, but it is also the source of systemic violence and injustice, because everything gets settled by confrontations of subjectivities, for want of objective, falsifiable criteria — and anyway, the end user is not even consulted. All those who dare point out the drifts, the abuses, the inconsistencies and the paradoxes are punished by the parish. The system reproduces itself by attracting (or keeping) only the same profiles as those who created the problem. All of this, once again, while making promises of emancipation that bind only those who believe them.

But all of this is only the visible ramifications of the underlying problem, and patching the problem on its social (or even societal) side the way one patches code, with small unit hacks, will fix nothing, neither in substance nor in form. Free software is to computer tooling what the public hospital is to the private clinic: everyone's public service, without distinction of anything. Not the public — but ultimately private — playground of a minority of idle programmers who can play computer between 5 and 9 p.m. while dinner cooks "all by itself".

In and outside open-source, design done well belongs to the domain of care. That is the conclusion I had been walking toward for years — long before the writing of this book gave it its name

— from searching for why I recognized myself less and less in free software and its ethic (the real one, not the one displayed and fantasized). The first computer program I ever wrote, in 2012, was called Scan2Thunderbird, and I wrote it for my grandfather who, having taken up computing at around 72 years old, could not manage to send a scanned attachment in an email. The application is a simple script that opens the required programs one after the other (the one to scan, page by page, then the one to send), with a reduced set of user options (limited to the common cases), and creates a new email with the scanned document already attached. All that remains is to write the message (with two fingers). No need to remember what to do, which program to use (what it is called and where it sits in the menu), or in what order: you are guided, step by step, by a single interface. That is design that reduces friction and eases pain, in the service of someone who would not have thought to ask for it.

It was not a family affair: my best contributions to free software have all had that shape, and none of them was a technical feat. Three examples, drawn from the same Darktable record.

For years, on the French Darktable forum, the same question kept coming back: how to enlarge the interface font, too small on recent screens. And for years, the same answer came down, from the maintainer and the regulars alike: quit the software, find a hidden configuration file, open it in a text editor, change `screen_dpi_overwrite` by hand — without getting the sign wrong[1]. No one saw a problem there: the question was being answered, again, instead of being eliminated. It wore me down. I ended up overhauling the interface so that everything — panels,

buttons, margins — follows the font size and the screen resolution; the work was rough, it was one of my first contributions in C, and the maintainer had to take it over to merge it[2]. The next year, the setting arrived in the preferences, one click away, and the question died out on the forum. The solution was simple, the impact enormous, and no one had thought of it: answering the same question ten times passed for peer help, when it is a design defect.

In November 2021, on the same forum, a colorblind user came to state his problem: a retoucher who cannot tell certain colors apart can shift a face from red to green without noticing, and he has no way to check it on his own — he has to ask a human with normal color vision: "here, what color is this? and there?". He was not asking for a miracle; his thread was titled "Guardrails or canes for the colorblind?": let the color picker say the colors in words. Two months later, I shipped a color glossary — a table linking chromaticity coordinates to names, completed with average skin tones measured across six populations — displayed in the sampling color picker[3]. It is, to this day, the only accessibility attempt in that software, and it was born from the request of a single user. That is not a bad ratio, it is the right one: one real, named, situated person is worth all the elastic "average users" of project discussions.

The rest of the field gives the measure of how exceptional this care is — which is to say, how abnormal. To this day, no other free photo editor — RawTherapee, ART, Filmulator, PhotoFlow, digiKam — has helped a colorblind person check the colors of his images. The best you find there is sympathy without follow-

through: at RawTherapee, in 2020, a colorblind user asks merely to be able to tell the mask previews apart; “Imho this is a request we should work on!”, replies the team — the ticket is still open six years later, with a single comment and zero lines of code. The worst is more candid: in 2014, on the same project, a contributor replies to the maintainer who was proposing accessible pictograms: “I am a bit puzzled why would cor blind person would need to do raw imaging” *(sic)*. As for the “colorblindness” filters of GIMP or G’MIC, they do the opposite of care: they simulate the impairment for the benefit of the normally sighted, to let them check their designs. No one equips the colorblind retoucher, because no one thinks of him — he is not in the room[4]. The argument is not technical: the 413 lines of code I put into Darktable for this purpose in 2022 are reusable practically as-is by any image editor written in C or C++. And they all are: RawTherapee, ART, Filmulator, PhotoFlow, digiKam, as well as GIMP, Krita, Siril or vkdt. Nothing to invent, nothing to fund: everything to copy; open-source is perfectly capable of that — when it feels like it, it proves it regularly. If nothing happened, it is purely a question of priorities: the problem is political.

There remain the orders of magnitude: these programs count their users in the millions. GIMP claims 50,000 downloads a day; Krita counted 5.6 million for the year 2021 alone; Darktable has served 8.5 million binary downloads, including 213,000 for the latest stable release in five weeks[5]. An almost exclusively male audience — 98% at Darktable, the only measurement available — where red-green colorblindness affects 8% of men: about one user in thirteen. Out of millions, that makes tens of thousands of

colorblind users left without a tool, at the very least. That is how many people are not in the room.

At the end of 2019, finally, I had opened a ticket at Darktable asking for packages built automatically every night: so that those who cannot compile could test the development version, report regressions before the release rather than after, or simply use it. The maintainer thought it was a good idea. Then nothing: the ticket rotted for three and a half years under the stale bot, the doctrine remaining "you can build yourself, it's not difficult". And the doctrine was paid for at the counter: at every release, users came to the tracker demanding updates of lagging third-party repositories, and were told that the project does not maintain packages[6]. In December 2022, I set up these nightly builds for Ansel — 70 hours of work, for a mechanical engineer by training who had to learn Linux and Windows packaging while doing it[7]. Two problems closed at once: testers without a compiler got their daily packages, and third-party repositories stopped being nobody's problem. Darktable ended up publishing its own nightly build — six months after mine, three and a half years after my request.

Three times the same shape: a pain deemed trivial by those who do not live it, a simple solution no one had taken seriously, a lasting effect. Care is not a virtue of character, it is a design method: it begins by holding the question from the other side of the counter worthy of being eliminated — not indefinitely answered.

Free software cannot remain the amusement of the few who can, at the expense of the majority who need. It must change, first its vision, and then its structure.

That will only happen by changing the personnel, because you cannot expect those who created the problem to solve it: for them, there is no problem. By abolishing intellectual and industrial property over software source code, they freed the expert user from the manufacturers' grip and gave him back control of his machine. And having freed the expert user, they believe they have freed everyone. The truth is they care only about themselves, and about those who look like them. A few hours of training and a new code of conduct will not turn the computer club into the public hospital of technicization: they are not there for that, and have no interest in it.

What must be done is to leave free software to its geeks, and invent public software: with the code free, but out of the hands of the free-software crowd. You cannot ask the privileged to think through the emancipation of the Other without falling, at best, into condescension.

---

1. Darktable.fr forum threads: "Taille de l'interface" https://Darktable.fr/t/taille-de-linterface/206 ("Interface size", 2016), "Police des outils" https://Darktable.fr/t/police-des-outils/831 ("Tool font", 2017-2018), "Taille des caractères" https://Darktable.fr/t/taille-des-caracteres/2005 ("Character size", February 2019), "Interface trop grande :(" https://Darktable.fr/t/interface-trop-grande/2489 ("Interface

too big :(", 2019-2020). In the second one, the maintainer Pascal Obry gives the procedure — edit `~/.config/Darktable/Darktablerc` and change `screen_dpi_overwrite` — specifying that you must "quit Darktable BEFORE making the change because Darktablerc is rewritten when Darktable exits" (« quitter Darktable AVANT de faire le changement car Darktablerc est réécrit lorsque Darktable se termine », our translation); the user had just gotten the sign of the value wrong. In February 2019, a regular observes: "there have already been several answers on this subject in the forum" (« il y a déjà eu plusieurs réponses sur ce sujet dans le forum », our translation). ↩

2. The overhaul: PR #2037 https://github.com/Darktable-org/Darktable/pull/2037 (January 2019, 132 comments), closed without merging — Pascal Obry takes it over, fixes it and merges it himself in PR #2381 https://github.com/Darktable-org/Darktable/pull/2381 (Darktable 3.0): "Merged in #2381 with a some fixes (redraw, accels). Thanks again for the hard work on this quite delicate issue Aurélien." The setting in the preferences arrives with Darktable 3.2: the font-size field by Chris Elston (PR #4974 https://github.com/Darktable-org/Darktable/pull/4974), the exposure of the DPI and scaling factors by me (PR #5469 https://github.com/Darktable-org/Darktable/pull/5469) — "it's long due and has been requested many many times", I wrote at the time. ↩

3. Thread “Gardes de fou ou cannes pour daltonien?” https://Darktable.fr/t/gardes-de-fou-ou-cannes-pour-daltonien/3834 (“Guardrails or canes for the colorblind?”, our translation), Darktable.fr, November 12, 2021. The author, a deutan trichromat: “I need human help to make sure the result has not betrayed my original intent: ‘here, what color is this? and there?’” (« j’ai besoin d’une aide humaine pour m’assurer que le résultat n’a pas trahi mon objectif initial : “ici, quelle est la couleur ? et là ?” », our translation). The code: commit of January 20, 2022 https://github.com/Darktable-org/Darktable/commit/dfe62a62211aa160420ca1c711749764ecfd7894 and PR #10935 https://github.com/Darktable-org/Darktable/pull/10935, merged manually by Pascal Obry after cleanup; shipped in Darktable 4.0 https://www.Darktable.org/2022/07/Darktable-4.0.0-released/ (July 2022) under the name “color glossary”: color names in the tooltip of the global color picker, and average skin tones for three body areas and six populations, after Xiao et al. (2017) and De Rigal et al. (2010). ↩

4. Audit of July 29, 2026 of the trackers, manuals and release notes of RawTherapee, ART, Filmulator, PhotoFlow, digiKam, GIMP, Krita, LightZone, vkdt, PhotoFlare, Fotoxx and Siril. RawTherapee 2020: request on the Local adjustments thread https://discuss.pixls.us/t/local-adjustments-is-in-dev/18795/24 — a developer’s reply: “Imho this is a request we should work on!” — then ticket #5807 https://github.com/RawTherapee/RawTherapee/issues/5807,

opened June 21, 2020, still open, a single comment. RawTherapee 2014: ticket #733 https://github.com/RawTherapee/RawTherapee/issues/733 — "I am a bit puzzled why would cor blind person would need to do raw imaging" (sic), in reply to the maintainer who was proposing to adopt the ColorADD pictograms. digiKam: no colorblindness-related report ever filed on bugs.kde.org. GIMP: the "Color Deficient Vision" https://docs.gimp.org/2.10/en/gimp-display-filter-dialog.html display filter, a simulation meant for the sighted; G'MIC: the `colorblind` command, "to check the accessibility of a web site or a graphical document". The field's only nuance: Siril (an astrophotography stacker) made the colors of its annotations configurable "For color-blind people" (2023) — an interface gesture, nothing about the colors of the image — while leaving the direct request of a colorblind user https://discuss.pixls.us/t/change-colour-of-background-extraction-points-for-colour-blind-users/36352 on its forum without any reply. On the other side, the colorblind users of Darktable's glossary testify to it, in 2024 https://discuss.pixls.us/t/Darktable-color-picker-color-names/45604 as in 2026 https://discuss.pixls.us/t/hue-value-to-color-convert/56595 : "It's like being without leg for years and you mounted me a new fresh prosthesis." Languages checked against repository statistics (July 29, 2026): RawTherapee, ART, Filmulator, PhotoFlow, digiKam, Krita and PhotoFlare are written in C++; GIMP, Siril and vkdt in C; G'MIC in C++. LightZone is a Java application, but its RAW decoding and part of its processing live in a native

C/C++ core (repository statistics: more C than Java) — and porting a 413-line lookup table can be done in any language. ↩

5. GIMP: "GIMP is downloaded over fifty thousand times a day and is used massively by millions of people across the world" (https://www.gimp.org/docs/userfaq.html, retrieved July 29, 2026), plus 3.7 million installs on the Flathub channel (Linux) alone. Krita: 5,637,579 downloads in 2021 https://krita.org/en/posts/2022/krita-in-2021-and-2022/, the last annual figure published — the project stopped publishing its statistics in 2023. Darktable: 8,550,205 cumulative binary downloads on GitHub (API, July 29, 2026), including 212,715 for version 5.6.0 in its first five weeks. Neither RawTherapee, nor digiKam, nor the others publish figures; downloads are not users, but these floors, measured channel by channel, are enough for the order of magnitude. The 98%: 25 women out of 1,243 answers, 2022 Darktable survey https://eng.aurelienpierre.com/2026/07/who-are-the-Darktable-users-in-2022/ (n = 1,253). The prevalence: about 8% of men and 0.4% of women of European descent, 4 to 6.5% of men in East Asia — J. Birch, "Worldwide prevalence of red-green color deficiency" https://doi.org/10.1364/JOSAA.29.000313, *Journal of the Optical Society of America A*, 29(3), 2012. The math: 0.98 × 8% + 0.02 × 0.4% ≈ 7.9%, or one user in thirteen. ↩

6. For example #5731 https://github.com/Darktable-org/Darktable/issues/5731 (2020) — reply: "The Darktable developers do not maintain distribution packages" — and

#11780 https://github.com/Darktable-org/Darktable/issues/11780 (2022): “The Darktable project does not maintain the OBS repo. It is maintained by the community and updated when the community has time to do so.” ↩

7. The request: #3925 “Provide an appimage ?” https://github.com/Darktable-org/Darktable/issues/3925, December 29, 2019 — favorably received by the maintainer (“Would be nice indeed !”), then the stale bot’s label, and closure on June 10, 2023: “We now have an AppImage in the nightly build.” The setup at Ansel: continuous preview release v0.0.0 https://github.com/aurelienpierreeng/ansel/releases/tag/v0.0.0 published on December 13, 2022, Linux AppImage pipeline on the 14th https://github.com/aurelienpierreeng/ansel/commit/3852b1ba2c0be84c5130d4c21844f3199b3624b2. The 70 hours: Ansel development diary, March 2023 https://ansel.photos/en/news/dev-diary-1/ — “I spent a total 70 h on making the nightly packages builds for Windows and Linux work for continuous delivery”, “something that Darktable never got right (“you can build yourself, it’s not difficult”)”. ↩

Annex — The measuring instruments

This book committed, from the prologue on, to being falsifiable. A figure that cannot be re-measured is not a measurement: it is an argument from authority with decimal places. This annex keeps the commitment. It publishes the instruments — the scripts that produce every measured series in the book — so that anyone with a computer and a spare hour can redo the counts, and contradict me with figures in hand. The chapter "Measuring without lying to yourself" stated the rule; here is the bench.

Three honest clarifications before the scripts. First, the book's measurements are dated snapshots — most from July-August 2026; the exact date sits in each footnote. Re-measuring today will give slightly different values, because the project is alive: between August 1 and August 4, 2026, the number of pull requests merged in 2025 returned by the GitHub API moved from 864 to 866. Two proposals requalified in three days change no conclusion; that is the order of magnitude of the noise, and it is stated. Second, these scripts are fair copies: the original measurements were made with equivalent commands over the course of the investigation, and their methods were then documented piece by piece; the scripts below reimplement those documented methods, and I verified that they reproduce the published series wherever a local check was possible. Third, one source is not public: the Matrix/IRC chat-room export. The counting script is published; the export is not — the notes of the

chapter "The closed circle" state its provenance and limits at every use.

You need Python 3, the `requests` module, `git`, and — for the GitHub scripts — a personal token in the `GITHUB_TOKEN` environment variable (the GraphQL API requires it; the free quota is ample). A local clone of the studied repository is needed for the first two scripts — a blobless clone is enough and weighs ten times less: `git clone --filter=blob:none https://github.com/Darktable-org/Darktable`.

The repository: demography and rhythm

This script produces the series of the chapters "Ship, don't dump" and "The committee will invent nothing": active contributors per year and per commitment tier (≥ 1, ≥ 5, ≥ 15 commits), median commits of the core (the scissors), the monthly pulse, contributor recurrence (how many years each one stays), the hourly heartbeat and the heat map. One detail matters: the hour is the author's, not the analyst's — `--date=format:` keeps the timezone recorded in each commit, where `format-local` would rewrite everything into the measurer's timezone.

```
#!/usr/bin/env python3
"""Demography and rhythm of a git repository, per year.
Usage: depot_demographie.py /path/to/clone"""
import subprocess, sys, collections
REPO = sys.argv[1]; SINCE = "2016-01-01"
def log(pretty, datefmt="short"):
    return subprocess.run(["git", "-C", REPO, "log", "--no-merges",
            f"--since={SINCE}", f"--date=format:{datefmt}",
            f"--pretty={pretty}"], capture_output=True,
    text=True).stdout.splitlines()

# 1) contributors per year, by tier (>=1, >=5, >=15 commits)
per = collections.defaultdict(collections.Counter)
for l in log("%ad|%aN|%aE", "%Y"):
    y, name, mail = l.split("|", 2)
    per[y][(name, mail.lower())] += 1
print("year  >=1  >=5  >=15  commits  median(core)")
for y in sorted(per):
    c = per[y]; core = sorted(n for n in c.values() if n >= 15)
    med = core[len(core)//2] if core else 0
    print(f"{y}  {len(c):4} {sum(1 for v in c.values() if v >= 5):4} "
          f"{len(core):4}  {sum(c.values()):6}  {med}")

```

```
# 2) recurrence: number of active years per contributor (2019+)
years = collections.defaultdict(set)
for l in log("%ad|%aN|%aE", "%Y"):
    y, name, mail = l.split("|", 2)
    if y >= "2019": years[(name, mail.lower())].add(y)
dist = collections.Counter(len(v) for v in years.values())
print("\nactive years -> contributors:", dict(sorted(dist.items())))

# 3) commits per month (the 10-year pulse)
mois = collections.Counter(log("%ad", "%Y-%m"))
print("\nper month:", dict(sorted(mois.items())))

# 4) heartbeat: hour + weekday in the AUTHOR's own timezone
#    (--date=format keeps each commit's recorded offset; format-local would
#     rewrite everything into the analyst's timezone — wrong for this use)
hours = collections.Counter(); days = collections.Counter()
for l in log("%ad", "%u %H"):
    d, h = l.split(); days[d] += 1; hours[h] += 1
print("\nby weekday (1=Mon):", dict(sorted(days.items())))
print("by hour:", dict(sorted(hours.items())))
```

Commit classification: who repairs what

The method behind the charts “Who repairs?” and “The mix does not change” of the chapter “Non-heroic ambition”: a commit is a fix (fix, bug, regression, crash, segfault, leak, closes/fixes #N) or a feature (add, new, introduce, implement, support, allow, enable, initial), by message keywords; a fix is a *patch-up* — the tax on recent work — when every `src/` file it touches was last modified by the same author within the threshold. Thresholds of 30, 90 and 180 days give identical results on Darktable; the remainder (refactoring, cleanup, build) is stated in the captions, because it is the dominant bucket. The limits are stated at the same place in the book: the classification reads messages, not diffs, and the last toucher of a file is not always the author of the defect — symmetric noise, insensitive to the threshold.

```
#!/usr/bin/env python3
"""Commit classification: true repair / self-patch / feature / other.
A FIX is a SELF-patch when every src/ file it touches was last modified
by the same author less than THRESHOLD days earlier (30/90/180 give
identical results on Darktable).
Usage: depot_classification.py /path/to/clone [threshold_days]"""
import subprocess, sys, re, datetime, collections
REPO = sys.argv[1]; THR = int(sys.argv[2]) if len(sys.argv) > 2 else 90
FIX = re.compile(r"\b(fix|bug|regression|crash|segfau
                 r"|(closes|fixes)\s+#\d+", re.I)
FEAT = re.compile(r"\b(add|new|introduce|implement|su
                  r"|enable|initial)\b", re.I)
raw = subprocess.run(["git", "-C", REPO, "log", "--no-merges",
    "--since=2016-01-01", "--date=unix", "--pretty=%x1e%H|%at|%ae|%s",
    "--name-only", "--", "src/"], capture_output=True, text=True).stdout
# \x1e (record separator) as delimiter: emails contain "@", subjects "|"...
commits = []
for block in raw.split("\x1e")[1:]:
    lines = block.strip().splitlines()
```

```
    h, t, ae, s = lines[0].split("|", 3)
    commits.append((h, int(t), ae, s, [f for f in lines[1:] if f]))
commits.sort(key=lambda c: c[1])  # chronological
last = {}  # file -> (author, epoch)
stats = collections.defaultdict(collections.Counter)
for h, t, ae, s, files in commits:
    y = datetime.datetime.fromtimestamp(t, datetime.timezone.utc).year
    cls = "other"
    if FIX.search(s):
        prev = [last.get(f) for f in files]
        selfp = files and all(p and p[0] == ae and (t - p[1]) < THR*86400
                              for p in prev)
        cls = "self-fix" if selfp else "ext-fix"
    elif FEAT.search(s):
        cls = "feat"
    stats[y][cls] += 1; stats[y]["total"] += 1
    for f in files: last[f] = (ae, t)
print("year  total  ext-fix  self-fix  feat  (rest=other)")
for y in sorted(stats):
    c = stats[y]
    print(f"{y}  {c['total']:5}  {c['ext-fix']:6}  {c['self-fix']:7}  {c['feat']:5}")
```

Pull requests: the veto and the concentration

The series of the chart "The veto dying out" ("Non-heroic ambition") and the principal merger's share ("Thinking against oneself"). Two method choices separate this from a naive count, and both are in the script. The merger is the `merged_by` field — the person who clicks the merge — not the author of the merge commit: counting merge commits inflates the figures with local branch merges and makes rebase merges invisible. And a proposal closed without merging whose comments say "manually merged" or "cherry-picked" is requalified as an acceptance — 211 cases out of 1,583 closures, concentrated in 2018-2020, which made the apparent 2018 refusal peak an artifact.

```
#!/usr/bin/env python3
"""Refusal rate + merger concentration, measured on pull requests.
Needs GITHUB_TOKEN. The merger = the merged_by field (who clicks the
merge), not merge-commit authorship. A closed-unmerged PR whose comments
say it was merged by hand is requalified as accepted."""
import os, requests, collections, re, time
API = "https://api.github.com/graphql"
H = {"Authorization": "bearer " + os.environ["GITHUB_TOKEN"]}
MANUAL = re.compile(r"manually merged|cherry.?picked"
                    r"|merged (this )?(by hand|manually)", re.I)
Q = """query($c:String) {repository(owner:"Darktable-org",name:"Darktable"){
 pullRequests(states:[MERGED,CLOSED],first:100,after:$c){
  pageInfo{hasNextPage endCursor}
  nodes{number state createdAt mergedAt mergedBy{login}
        comments(last:20){nodes{bodyText}}}}}"""
merged = collections.defaultdict(collections.Counter)
closed = collections.Counter(); requal = collections.Counter(); cur = None
while True:
```

```
    r = requests.post(API, json={"query": Q,
"variables": {"c": cur}},
                      headers=H).json()
    pr = r["data"]["repository"]
["pullRequests"]
    for n in pr["nodes"]:
        if n["state"] == "MERGED":
            who = n["mergedBy"]["login"] if
n["mergedBy"] else "?"
            merged[n["mergedAt"][:4]][who] +=
1
        else:
            y = n["createdAt"][:4]
            if
any(MANUAL.search(c["bodyText"])
                   for c in n["comments"]
["nodes"]):
                requal[y] += 1
            else:
                closed[y] += 1
    if not pr["pageInfo"]["hasNextPage"]:
break
    cur = pr["pageInfo"]["endCursor"];
time.sleep(0.5)
print("year  merged  refused  requalified
refusal%  top-merger-share")
for y in sorted(merged):
    m = sum(merged[y].values()); ref =
closed[y]
    top = max(merged[y].values()) if
merged[y] else 0
    print(f"{y}  {m:5}  {ref:5}
{requal[y]:3}  "
```

```
            f"{100*ref/(m+ref+requal[y]):5.1f}
    {100*top/m:5.1f}")
```

Issues: closure capacity and the testers

The series of “The decline of closure capacity”, “The end of life of Darktable issues” and “The testers who leave” (“Ship, don’t dump”, “The committee will invent nothing”): who closes — a human or the stale bot — is read from the actor of the closing event; regular reporters (≥ 3 issues in the year) and their attrition are counted on authors. The book’s caution applies to everything that follows: a closure is not a repair, and the script does not claim to judge what a human rightly closed.

```
#!/usr/bin/env python3
"""Issues: creation, closure capacity (human vs stale bot), reporters.
Needs GITHUB_TOKEN. The closer is the actor of the CLOSED_EVENT; the
stale bot answers to the github-actions login."""
import os, requests, collections, time
API = "https://api.github.com/graphql"; BOTS = {"github-actions"}
H = {"Authorization": "bearer " + os.environ["GITHUB_TOKEN"]}
Q = """query($c:String) {repository(owner:"Darktable-org",name:"Darktable"){
 issues(first:100,after:$c) {pageInfo{hasNextPage endCursor}
  nodes{createdAt closedAt author{login}
   timelineItems(itemTypes:[CLOSED_EVENT],last:1){nodes{
    ... on ClosedEvent{actor{login}}}}}}}}"""
created = collections.Counter(); human = collections.Counter()
bot = collections.Counter(); cur = None
authors = collections.defaultdict(collections.Counter)
while True:
    r = requests.post(API, json={"query": Q, "variables": {"c": cur}},
                      headers=H).json()
    ii = r["data"]["repository"]["issues"]
    for n in ii["nodes"]:
```

```
        y = n["createdAt"][:4]; created[y] += 1
        if n["author"]: authors[y][n["author"]["login"]] += 1
        if n["closedAt"]:
            ev = n["timelineItems"]["nodes"]
            actor = ev[0]["actor"]["login"] if ev and ev[0].get("actor") else "?"
            target = bot if actor in BOTS or actor.endswith("[bot]") else human
            target[n["closedAt"][:4]] += 1
    if not ii["pageInfo"]["hasNextPage"]: break
    cur = ii["pageInfo"]["endCursor"]; time.sleep(0.5)
print("year  created  closed-by-human  closed-by-bot  regulars(>=3)  gone")
prev = set()
for y in sorted(created):
    reg = {a for a, n in authors[y].items() if n >= 3}
    gone = len(prev - reg) if prev else ""
    print(f"{y}  {created[y]:5}  {human[y]:5}  {bot[y]:5}  {len(reg):4}  {gone}")
    prev = reg
```

The forum: the threads, and how it names me

Discourse serves everything as JSON — just append `.json` to the URLs. This script counts the threads opened per year in the Darktable category of discuss.pixls.us ("The decoupling") and the quarterly occurrences of the naming forms ("How the forum names me", "The closed circle"). The limit is stated in the chart's footnote: Discourse search caps its results; quarterly windows keep the counts exact for rare terms, and the aggregate total comes from `grouped_search_result`.

```
#!/usr/bin/env python3
"""discuss.pixls.us: threads per year in a category; naming forms per
quarter. Discourse serves JSON everywhere; no token needed, but paced."""
import requests, time, collections
BASE = "https://discuss.pixls.us"
def get(u):
    time.sleep(0.6)
    return requests.get(BASE + u, timeout=30).json()

# 1) threads opened per year in the Darktable category (id 19, /site.json)
topics = collections.Counter(); page = 0
while True:
    d = get(f"/c/Darktable/19.json?page={page}")
    tl = d["topic_list"]["topics"]
    if not tl: break
    for t in tl: topics[t["created_at"][:4]] += 1
    page += 1
print("threads/year:", dict(sorted(topics.items())))

# 2) naming study: occurrences of each form, per quarter
for term in ('"Aurélien"', '"AP"'):
    for y in range(2019, 2027):
        for q, (a, b) in enumerate([("01-01", "04-01"), ("04-01", "07-01"),
```

```
                                        ("07-01",
"10-01"), ("10-01", "12-31")], 1):
            d = get(f"/search.json?q=
{term}%20after:{y}-{a}%20before:{y}-{b}")
            n =
d.get("grouped_search_result",
{}).get("post_count",
                    len(d.get("posts",
[])))
            print(f"{term} {y}-Q{q}: {n}")
```

The Matrix chat room: counting on the export

The chat-room export is private — it is the book's only source the reader cannot re-measure, and the footnotes that rely on it say so at every use. The counting script is published as instrument honesty: it states exactly what was counted (messages of type `m.room.message`, per sender, per quarter). The fine-grained codings — separating addressed peer help from small talk — were done by hand on this output, and are described in the footnotes that use them.

```python
#!/usr/bin/env python3
"""Element/Matrix room export (JSON): messages per sender, per quarter.
The export is PRIVATE (names live as roles in the book); this script
documents the counting, it cannot re-fetch the data.
Usage: salon_matrix.py export.json [sender_id]"""
import json, sys, datetime, collections
d = json.load(open(sys.argv[1]))
who = sys.argv[2] if len(sys.argv) > 2 else None
per = collections.Counter(); senders = collections.Counter()
for m in d["messages"]:
    if m.get("type") != "m.room.message": continue
    t = datetime.datetime.fromtimestamp(m["origin_serv

datetime.timezone.utc)
    key = f"{t.year}-Q{(t.month-1)//3+1}"
    senders[m["sender"]] += 1
    if not who or m["sender"] == who: per[key] += 1
print("top senders:", senders.most_common(10))
print("per quarter:", dict(sorted(per.items())))
```

From series to charts

The book's charts are JSON files in the Plotly format ("data" + "layout"), read by a Hugo shortcode that carries the title and the caption. The template:

```python
#!/usr/bin/env python3
"""Series -> the book's chart JSON ({"data":[...],"layout":{...}}),
read by the Hugo plotly shortcode. Title and caption live in the
shortcode call, not in the JSON."""
import json
def chart(series, xtitle="", ytitle="", kind="scatter", out="chart.json"):
    data = [{"type": kind, "name": name, "x": x, "y": y}
            for name, (x, y) in series.items()]
    layout = {"xaxis": {"title": {"text": xtitle}},
              "yaxis": {"title": {"text": ytitle}},
              "margin": {"l": 60, "r": 20, "t": 10, "b": 40},
              "legend": {"orientation": "h"}}
    json.dump({"data": data, "layout": layout}, open(out, "w"),
              ensure_ascii=False, indent=1)
if __name__ == "__main__":
    chart({"demo": ([2019, 2020, 2021], [1, 4, 2])}, "year", "value")
```

The three Brooks charts (“The committee will invent nothing”) mine nothing: they are analytic curves — n(n−1)/2 channels for n people — drawn with this template. The exhaustion chart (“The invisible worker”) combines six personal series produced by the scripts above — commits, opened pull requests, reviews, issue

triage, messages on the two forums — each normalized to its own peak; and the chart "The empty house" crosses three sources (GitHub issues, repository contributors, chat-room peer help) with the corresponding scripts, excluding my own contributions where the caption says so.

There are the instruments. They fit in three hundred lines, and that is the point: none of this book's measurements requires a laboratory. They only required looking at the records the project itself keeps, publicly, and has kept for ten years — and accepting to write down what one finds there.

Contents

Prologue — Three paths

Part I — What designing means

1. The word and the thing
2. There is no technical problem
3. Listening without obeying

Part II — The discipline

4. The root and the symptom
5. Stating the problem
6. Measuring without lying to yourself
7. Counting the absent
8. Decide, then hold
9. Tooling up without submitting

Part III — The machine

10. Beneath the interface, dragons
11. In praise of the simple
12. Code is a liability
13. The cargo cult
14. Intelligence upstream
15. The transparent tool
16. The instrument and the machine
17. Enabling without holding the brush

18. Finishing and maintaining

Part IV — The collective

19. Ship, don't dump
20. The committee will invent nothing
21. The invisible worker
22. Working for the work
23. Non-heroic ambition

Part V — Freedom and its conditions

24. Freedom in a void
25. What insularity produces
26. Three-tiered violence
27. The closed circle
28. The well-tempered conflict
29. Thinking against oneself
30. The incompetent worker
31. The pact

Conclusion — Taking care

Epilogue — Behind every problem, a human

Annex — The measuring instruments